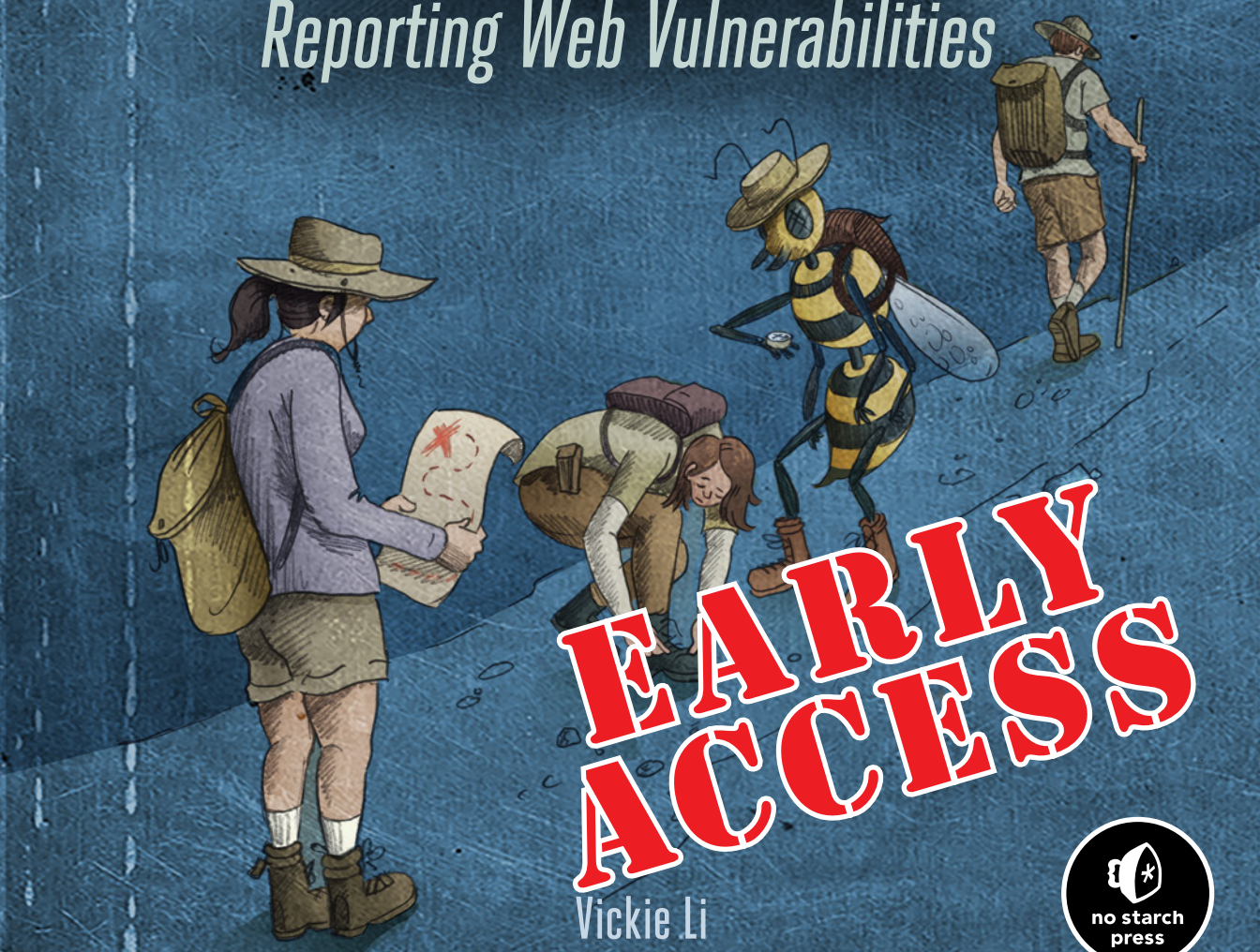
Bug Bounty Bootcamp

The Guide to Finding and Reporting Web Vulnerabilities



EARLY ACCESS

Vickie Li

no starch press

NO STARCH PRESS EARLY ACCESS PROGRAM: FEEDBACK WELCOME!

Welcome to the Early Access edition of the as yet unpublished *Bug Bounty Bootcamp* by Vickie Li! As a prepublication title, this book may be incomplete and some chapters may not have been proofread.

Our goal is always to make the best books possible, and we look forward to hearing your thoughts. If you have any comments or questions, email us at **earlyaccess@nostarch.com**. If you have specific feedback for us, please include the page number, book title, and edition date in your note, and we'll be sure to review it. We appreciate your help and support!

We'll email you as new chapters become available. In the meantime, enjoy!

BUG BOUNTY BOOTCAMP

VICKIE LI

Early Access edition, 5/5/21

ISBN-10: 978-1-7185-0154-6 (print)
ISBN-13: 978-1-7185-0155-3 (ebook)

Publisher: William Pollock
Production Manager: Rachel Monaghan
Production Editors: Miles Bond and Dapinder Dosanjh
Developmental Editor: Frances Saux
Cover Design: Rick Reese
Interior Design: Octopod Studios
Technical Reviewer: Aaron Guzman
Copyeditor: Sharon Wilkey
Compositor: Jeff Lytle, Happenstance Type-O-Rama

CONTENTS

The chapters in **red** are included in this Early Access PDF.

I

THE INDUSTRY

1

PICKING A BUG BOUNTY PROGRAM

Bug bounty programs: are they all the same? Finding the right program to target is the first step to becoming a successful bug bounty hunter. Many programs have emerged within the past few years, and it's difficult to figure out which ones will provide the best monetary rewards, experience, and learning opportunities.

A *bug bounty program* is an initiative in which a company invites hackers to attack its products and service offerings. So how should you pick a program? And how should you prioritize their different metrics, such as the asset types involved, whether the program is hosted on a platform, whether it's public or private, the program's scope, the payout amounts, and response times?

In this chapter, we'll explore types of bug bounty programs, analyze the benefits and drawbacks of each, and figure out which one you should go for.

The State of the Industry

Bug bounties are currently one of the most popular ways for organizations to receive feedback about security bugs. Large corporations, like PayPal and Facebook, as well as government agencies like the US Department of Defense, have all embraced the idea. Yet not too long ago, reporting a vulnerability to a company would have more likely landed you in jail than gotten you a reward.

In 1995, Netscape launched the first-ever bug bounty program. The company encouraged users to report bugs found in its brand-new browser, the Netscape Navigator 2.0, introducing the idea of crowdsourced security testing to the internet world. Mozilla launched the next corporate bug bounty program nine years later, in 2004, inviting users to identify bugs in the Firefox browser.

But it was not until the 2010s that offering bug bounties become a popular practice. That year, Google launched its program, and Facebook followed suit in 2011. These two programs kick-started the trend of using bug bounties to augment a corporation's in-house security infrastructure.

As bug bounties became a more well-known strategy, bug-bounty-as-a-service *platforms* emerged. These platforms help companies set up and operate their programs. For example, they provide a place for companies to host their programs, a way to process reward payments, and a centralized place to communicate with bug bounty hunters.

The two largest of these platforms, HackerOne and Bugcrowd, both launched in 2012. After that, a few more platforms, such as Synack, Cobalt, and Intigriti, came to the market. These platforms and managed bug bounty services allow even companies with limited resources to run a security program. Today, large corporations, small startups, nonprofits, and government agencies alike have adopted bug bounties as an additional security measure and a fundamental piece of their security policies. You can read more about the history of bug bounty programs at https://en.wikipedia.org/wiki/Bug_bounty_program.

The term *security program* usually refers to information security policies, procedures, guidelines, and standards in the larger information security industry. In this book, I use *program* or *bug bounty program* to refer to a company's bug bounty operations. Today, tons of programs exist, all with their unique characteristics, benefits, and drawbacks. Let's examine these.

Asset Types

In the context of a bug bounty program, an *asset* is an application, website, or product that you can hack. There are different types of assets, each with its own characteristics, requirements, and pros and cons. After considering these differences, you should choose a program with assets that play to your strengths, based on your skill set, experience level, and preferences.

Social Sites and Applications

Anything labeled *social* has a lot of potential for vulnerabilities, because these applications tend to be complex and involve a lot of interaction among users, and between the user and the server. That's why the first type of bug bounty program we'll talk about targets social websites and applications. The term *social application* refers to any site that allows users to interact with each other. Many programs belong to this category: examples include the bug bounty program for HackerOne and programs for Facebook, Twitter, GitHub, and LINE.

Social applications need to manage interactions among users, as well as each user's roles, privileges, and account integrity. They are typically full of potential for critical web vulnerabilities such as insecure direct object references (IDORs), info leaks, and account takeovers. These vulnerabilities occur when many users are on a platform, and when applications mismanage user information; when the application does not validate a user's identity properly, malicious users can assume the identity of others.

These complex applications also often provide a lot of user input opportunities. If input validation is not performed properly, these applications are prone to injection bugs, like SQL injection (SQLi) or cross-site scripting (XSS).

If you are a newcomer to bug bounties, I recommend that you start with social sites. The large number of social applications nowadays means that if you target social sites, you'll have many programs to choose from. Also, the complex nature of social sites means that you'll encounter a vast attack surface with which to experiment. (An application's *attack surface* refers to all of the application's different points that an attacker can attempt to exploit.) Finally, the diverse range of vulnerabilities that show up on these sites means that you will be able to quickly build a deep knowledge of web security.

The skill set you need to hack social programs includes the ability to use a proxy, like the Burp Suite proxy introduced in Chapter 4, and knowledge about web vulnerabilities such as XSS and IDOR. You can learn more about these in Chapters 6 and 10. It's also helpful to have some JavaScript programming skills and knowledge about web development. However, these skills aren't required to succeed as a hacker.

But these programs have a major downside. Because of the popularity of their products and the low barrier of entry, they're often very competitive and have many hackers hunting on them. Social media platforms such as Facebook and Twitter are some of the most targeted programs.

General Web Applications

General web applications are also a good target for beginners. Here, I am referring to any web applications that do not involve user-to-user interaction. Instead, users interact with the server to access the application's features. Targets that fall into these categories can include static websites, cloud applications, consumer services like banking sites, and web portals of Internet of Things (IoT) devices or other connected hardware. Like social sites, they

are also quite diverse and lend themselves well to a variety of skill levels. Examples include the programs for Google, the US Department of Defense, and Credit Karma.

That said, in my experience, they tend to be a little more difficult to hack than social applications, and their attack surface is smaller. If you're looking for account takeovers and info leak vulnerabilities, you won't have as much luck because there aren't a lot of opportunities for users to interact with others and potentially steal others' information. The types of bugs that you'll find in these applications are slightly different. You'll need to look for server-side vulnerabilities and vulnerabilities specific to the application's technology stack. You could also look for commonly found network vulnerabilities, like subdomain takeovers. This means you'll have to know about both client-side and server-side web vulnerabilities, and you should have the ability to use a proxy. It's also helpful to have some knowledge about web development and programming.

These programs can range in popularity. However, most of them have a low barrier of entry, so you can most likely get started hacking right away!

Mobile Applications (Android, iOS, and Windows)

After you get the hang of hacking web applications, you may choose to specialize in *mobile applications*. Mobile programs are becoming prevalent; after all, most web apps have a mobile equivalent nowadays. They include programs for Facebook Messenger, the Twitter app, the LINE mobile app, the Yelp app, and the Gmail app.

Hacking mobile applications requires the skill set you've built from hacking web applications, as well as additional knowledge about the structure of mobile apps and programming techniques related to the platform. You should understand attacks and analysis strategies like certificate pinning bypass, mobile reverse engineering, and cryptography.

Hacking mobile applications also requires a little more setup than hacking web applications, as you'll need to own a mobile device that you can experiment on. A good mobile testing lab consists of a regular device, a rooted device, and device emulators for both Android and iOS. A *rooted device* is one for which you have admin privileges. It will allow you to experiment more freely, because you can bypass the mobile system's safety constraints. An *emulator* is a virtual simulation of mobile environments that you run on your computer. It allows you to run multiple device versions and operating systems without owning a device for each setup.

For these reasons, mobile applications are less popular among bug bounty hunters than web applications. However, the higher barrier of entry for mobile programs is an advantage for those who do participate. These programs are less competitive, making it relatively easy to find bugs.

APIs

Application programming interfaces (APIs) are specifications that define how other applications can interact with an organization's assets, such as to retrieve or alter their data. For example, another application might be able

to retrieve an application's data via HyperText Transfer Protocol (HTTP) messages to a certain endpoint, and the application will return data in the format of Extensible Markup Language (XML) or JavaScript Object Notation (JSON) messages.

Some programs put a heightened focus on API bugs in their bug bounty programs if they're rolling out a new version of their API. A secure API implementation is key to preventing data breaches and protecting customer data. Hacking APIs requires many of the same skills as hacking web applications, mobile applications, and IoT applications. But when testing APIs, you should focus on common API bugs like data leaks and injection flaws.

Source Code and Executables

If you have more-advanced programming and reversing skills, you can give *source code* and *executable programs* a try. These programs encourage hackers to find vulnerabilities in an organization's software by directly providing hackers with an open sourced codebase or the binary executable. Examples include the Internet Bug Bounty, the program for the PHP language, and the WordPress program.

Hacking these programs can entail analyzing the source code of open source projects for web vulnerabilities and fuzzing binaries for potential exploits. You usually have to understand coding and computer science concepts to be successful here. You'll need knowledge of web vulnerabilities, programming skills related to the project's codebase, and code analysis skills. Cryptography, software development, and reverse engineering skills are helpful.

Source code programs may sound intimidating, but keep in mind that they're diverse, so you have many to choose from. You don't have to be a master programmer to hack these programs; rather, aim for a solid understanding of the project's tech stack and underlying architecture. Because these programs tend to require more skills, they are less competitive, and only a small proportion of hackers will ever attempt them.

Hardware and IoT

Last but not least are hardware and IoT programs. These programs ask you to hack devices like cars, smart televisions, and thermostats. Examples include the bug bounty programs of Tesla and Ford Motor Company.

You'll need highly specific skills to hack these programs: you'll often have to acquire a deep familiarity with the type of device that you're hacking, in addition to understanding common IoT vulnerabilities. You should know about web vulnerabilities, programming, code analysis, and reverse engineering. Also, study up on IoT concepts and industry standards such as digital signing and asymmetric encryption schemes. Finally, cryptography, wireless hacking, and software development skills will be helpful too.

Although some programs will provide you with a free device to hack, that often applies to only the select hackers who've already established a relationship with the company. To begin hacking on these programs, you might need the funds to acquire the device on your own.

Since these programs require specialized skills and a device, they tend to be the least competitive.

Bug Bounty Platforms

Companies can host bug bounty programs in two ways: bug bounty platforms and independently hosted websites.

Bug bounty platforms are websites through which many companies host their programs. Usually, the platform directly awards hackers with reputation points and money for their results. Some of the largest bug bounty platforms are HackerOne, Bugcrowd, Intigriti, Synack, and Cobalt.

Bug bounty platforms are an intermediary between hackers and security teams. They provide companies with logistical assistance for tasks like payment and communication. They also often offer help managing the incoming reports by filtering, deduplicating, and triaging bug reports for companies. Finally, these platforms provide a way for companies to gauge a hacker's skill level via hacker statistics and reputation. This allows companies that do not wish to be inundated with low-quality reports to invite experienced hackers to their private programs. Some of these platforms also screen or interview hackers before allowing them to hack on programs.

From the hacker's perspective, bug bounty platforms provide a centralized place to submit reports. They also offer a seamless way to get recognized and paid for your findings.

On the other hand, many organizations host and manage their bug bounty programs without the help of platforms. Companies like Google, Facebook, Apple, and Medium do this. You can find their bug bounty policy pages by visiting their websites, or by searching "*Company_Name* bug bounty program" online.

As a bug bounty hunter, should you hack on a bug bounty platform? Or should you go for companies' independently hosted programs?

The Pros . . .

The best thing about bug bounty platforms is that they provide a lot of transparency into a company's process, because they post disclosed reports, metrics about the programs' triage rates, payout amounts, and response times. Independently hosted programs often lack this type of transparency. In the bug bounty world, *triage* refers to the confirmation of vulnerability.

You also won't have to worry about the logistics of emailing security teams, following up on reports, and providing payment and tax info every time you submit a vulnerability report. Bug bounty programs also often have reputation systems that allow you to showcase your experience so you can gain access to invite-only bug bounty programs.

Another pro of bug bounty platforms is that they often step in to provide conflict resolution and legal protection as a third party. If you submit a report to a non-platform program, you have no recourse in the final bounty decision.

Ultimately, you can't always expect companies to pay up or resolve reports in the current state of the industry, but the hacker-to-hacker feedback system that platforms provide is helpful.

. . . and the Cons

However, some hackers avoid bug bounty platforms because they dislike how those platforms deal with reports. Reports submitted to platform-managed bug bounty programs get handled by *triagers*, third-party employees who often aren't familiar with all the security details about a company's product. Complaints about triagers handling reports improperly are common.

Programs on platforms also break the direct connection between hackers and developers. With a direct program, you often get to discuss the vulnerability with a company's security engineers, making for a great learning experience.

Finally, public programs on bug bounty platforms are often crowded, because the platform gives them extra exposure. On the other hand, many privately hosted programs don't get as much attention from hackers and are thus less competitive. And for the many companies that do not contract with bug bounty platforms, you have no choice but to go off platforms if you want to participate in their programs.

Scope, Payouts, and Response Times

What other metrics should you consider when picking a program, besides its asset types and platform? On each bug bounty program's page, metrics are often listed to help you assess the program. These metrics give insight into how easily you might be able to find bugs, how much you might get paid, and how well the program operates.

Program Scope

First, consider the scope. A program's *scope* on its policy pages specifies what and how you are allowed to hack. There are two types of scopes: asset and vulnerability. The *asset scope* tells you which subdomain, products, and applications you can hack. And the *vulnerability scope* specifies which vulnerabilities the company will accept as valid bugs.

For example, the company might list the subdomains of its website that are in and out of scope:

In-scope assets	**Out-of-scope assets**
a.example.com	*dev.example.com*
b.example.com	*test.example.com*
c.example.com	
users.example.com	
landing.example.com	

Assets that are listed as in scope are the ones that you are allowed to hack. On the other hand, assets that are listed as out of scope are off-limits to bug bounty hunters. Be extra careful and abide by the rules! Hacking an out-of-scope asset is illegal.

The company will also often list the vulnerabilities it considers valid bugs:

In-scope vulnerabilities	Out-of-scope vulnerabilities
All except the ones listed as out of scope	Self-XSS
	Clickjacking
	Missing HTTP headers and other best practices without direct security impact
	Denial-of-service attacks
	Use of known-vulnerable libraries, without proof of exploitability
	Results of automated scanners, without proof of exploitability

The out-of-scope vulnerabilities that you see in this example are typical of what you would find in bug bounty programs. Notice that many programs consider non-exploitable issues, like violations of best practice, to be out of scope.

Any program with large asset and vulnerability scopes is a good place to start for a beginner. The larger the asset scope, the larger the number of target applications and web pages you can look at. When a program has a big asset scope, you can often find obscure applications that are overlooked by other hackers. This typically means less competition when reporting bugs.

And the larger the vulnerability scope, the more types of bugs the organization is willing to hear reports about. These programs are a lot easier to find bugs in, because you have more opportunities, and so can play to your strengths.

Payout Amounts

The next metric you should consider is the program's *payout amounts*. When it comes to payments, you can distinguish between two types of programs: vulnerability disclosure programs and bug bounty programs. Vulnerability disclosure programs (VDPs) don't pay monetary rewards, and bug bounty programs do.

VDPs, or *reputation-only programs*, do not pay for findings but often offer rewards such as reputation points and swag. They are a great way to learn about hacking if making money is not your primary objective. Since they don't pay, they're less competitive, and so easier to find bugs in. You can use them to practice finding common vulnerabilities and communicating with security engineers.

On the other hand, bug bounty programs offer varying amounts of monetary rewards for your findings. In general, the more severe the vulnerability, the more the report will pay. But different programs have a different payout

average for each level of severity. You can find a program's payout information on its bug bounty pages, usually listed in a section called the *payout table*. Typically, low-impact issues will pay anywhere from $50 to $500 (USD), while critical issues can pay upward of $10,000. However, the bug bounty industry is evolving, and payout amounts are increasing for high-impact bugs. For example, Apple now rewards up to $1 million for the most severe vulnerabilities.

Response Time

Finally, consider the program's average *response time*. Some companies will handle and resolve your reports within a few days, while others take weeks or even months to finalize their fixes. Delays often happen because of the security team's internal constraints, like a lack of personnel to handle reports, a delay in issuing security patches, and a lack of funds to timely reward researchers. Sometimes, delays happen because researchers have sent bad reports without clear reproduction steps.

Prioritize programs with fast response times. Waiting for responses from companies can be a frustrating experience, and when you first start, you're going to make a lot of mistakes. You might misjudge the severity of a bug, write an unclear explanation, or make technical mistakes in the report. Rapid feedback from security teams will help you improve, and turn you into a competent hacker faster.

Private Programs

Most bug bounty platforms distinguish between public and private programs.

Public programs are those that are open to all; anyone can hack and submit bugs to these programs, as long as they abide by the laws and the bug bounty program's policies.

On the other hand, *private programs* are open to only invited hackers. For these, companies ask hackers with a certain level of experience and a proven track record to attack the company and submit bugs to it. Private programs are a lot less competitive than public ones because of the limited number of hackers participating. Therefore, it's much easier to find bugs in them. Private programs also often have a much faster response time, because they receive fewer reports on average.

Participating in private programs can be extremely advantageous. But how do you get invited to one? Figure 1-1 shows a private invitation notification on the HackerOne platform.

Figure 1-1: A private invitation notification on the HackerOne platform. When you hack on a bug bounty platform, you can often get invites to the private programs of different companies.

Companies send private invites to hackers who have proven their abilities in some way, so getting invites to private programs isn't difficult once

you've found a couple of bugs. Different bug bounty platforms will have different algorithms to determine who gets the invites, but here are some tips to help you get there.

First, submit a few bugs to public programs. To get private invites, you often need to gain a certain number of reputation points on a platform, and the only way to begin earning these is to submit valid bugs to public programs. You should also focus on submitting high-impact vulnerabilities. These vulnerabilities will often reward you with higher reputation points and help you get private invites faster. In each of the chapters in Part II of this book, I make suggestions for how you can escalate the issues you discover to craft the highest-impact attacks. On some bug bounty platforms, like HackerOne, you can also get private invites by completing tutorials or solving Capture the Flag (CTF) challenges.

Next, don't spam. Submitting nonissues often causes a decrease in reputation points. Most bug bounty platforms limit private invites to hackers with points above a certain threshold.

Finally, be polite and courteous when communicating with security teams. Being rude or abusive to security teams will probably get you banned from the program and prevent you from getting private invites from other companies.

Choosing the Right Program

Bug bounties are a great way to gain experience in cybersecurity and earn extra bucks. But the industry has been getting more competitive. As more people are discovering these programs and getting involved in hacking on them, it's becoming increasingly difficult for beginners to get started. That's why it's important to pick a program that you can succeed in from the very start.

Before you develop a bug hunter's intuition, you often have to rely on low-hanging fruit and well-known techniques. This means many other hackers will be able to find the same bugs, often much faster than you can. So, it's a good idea to pick a program that more experienced bug hunters pass over to avoid competition. You can find these underpopulated programs in two ways: look for unpaid programs or go for programs with big scopes.

Try going for vulnerability disclosure programs first. Unpaid programs are often ignored by experience bug hunters, since they don't pay monetary rewards. But they still earn you points and recognition! And that recognition might be just what you need to get an invite to a private, paid program.

Picking a program with a large scope means you'll be able to look at a larger number of target applications and web pages. This dilutes the competition, as fewer hackers will report on any single asset or vulnerability type. Go for programs with fast response times to prevent frustration and get feedback as soon as possible.

One last thing that you can incorporate into your decision process is the reputation of the program. If you can, gather information about a

company's process through its disclosed reports and learn from other hackers' experiences. Does the company treat its reporters well? Are they respectful and supportive? Do they help you learn? Pick programs that will be supportive while you are still learning, and programs that will reward you for the value that you provide.

Choosing the right program for your skill set is crucial if you want to break into the world of bug bounties. This chapter should have helped you sort out the various programs that you might be interested in. Happy hacking!

A Quick Comparison of Popular Programs

After you've identified a few programs that you are interested in, you could list the properties of each one to compare them. In Table 1-1, let's compare a few of the popular programs introduced in this chapter.

Table 1-1: A Comparison of Three Bug Bounty Programs: HackerOne, Facebook, and GitHub

Program	Asset type	In scope	Payout amount	Response time
HackerOne	Social site	*https://hackerone.com* *https://api.hackerone.com* **.vpn.hackerone.net* *https://www.hackerone.com* And more assets . . . Any vulnerability except exclusions are in scope.	$500–$15,000+ USD	Fast. Average time to response is 5 hours. Average time to triage is 15 hours.
Facebook	Social site, nonsocial site, mobile site, IoT, and source code	Instagram Internet.org / Free Basics Oculus Workplace Open source projects by Facebook WhatsApp Portal FBLite Express Wi-Fi Any vulnerability except exclusions are in scope.	$500 minimum	Based on my experience, pretty fast!
GitHub	Social site	*https://blog.github.com* *https://community.github.com/* *http://resources.github.com/* And more assets . . . Use of known-vulnerable software. Clickjacking a static site. Including HTML in Markdown content. Leaking email addresses via .patch links. And more issues . . .	$617–$30,000 USD	Fast. Average time to response is 11 hours. Average time to triage is 23 hours.

2

SUSTAINING YOUR SUCCESS

Even if you understand the technical information in this book, you may have difficulty navigating the nuances of bug bounty programs. Or you might be struggling to actually locate legitimate bugs and aren't sure why you're stuck. In this chapter, we'll explore some of the factors that go into making a successful bug bounty hunter. We'll cover how to write a report that properly describes your findings to the security team, build lasting relationships with the organizations you work with, and overcome obstacles during your search for bugs.

Writing a Good Report

A bug bounty hunter's job isn't just finding vulnerabilities; it's also explaining them to the organization's security team. If you provide a well-written report, you'll help the team you're working with reproduce the exploit, assign it to the appropriate internal engineering team, and fix the issue faster. The faster a vulnerability is fixed, the less likely malicious hackers are to exploit it. In this section, I'll break down the components of a good vulnerability report and introduce some tips and tricks I've learned along the way.

Step 1: Craft a Descriptive Title

The first part of a great vulnerability report is always a descriptive title. Aim for a title that sums up the issue in one sentence. Ideally, it should allow the security team to immediately get an idea of what the vulnerability is, where it occurred, and its potential severity. To do so, it should answer the following questions: What is the vulnerability you've found? Is it an instance of a well-known vulnerability type, such as IDOR or XSS? Where did you find it on the target application?

For example, instead of a report title like "IDOR on a Critical Endpoint," use one like "IDOR on *https://example.com/change_password* Leads to Account Takeover for All Users." Your goal is to give the security engineer reading your report a good idea of the content you'll discuss in the rest of it.

Step 2: Provide a Clear Summary

Next, provide a report summary. This section includes all the relevant details you weren't able to communicate in the title, like the HTTP request parameters used for the attack, how you found it, and so on.

Here's an example of an effective report summary:

> The *https://example.com/change_password* endpoint takes two POST body parameters: `user_id` and `new_password`. A POST request to this endpoint would change the password of user `user_id` to `new_password`. This endpoint is not validating the `user_id` parameter, and as a result, any user can change anyone else's password by manipulating the `user_id` parameter.

A good report summary is clear and concise. It contains all the information needed to understand a vulnerability, including what the bug is, where the bug is found, and what an attacker can do when it's exploited.

Step 3: Include a Severity Assessment

Your report should also include an honest assessment of the bug's severity. In addition to working with you to fix vulnerabilities, security teams have other responsibilities to tend to. Including a severity assessment will help them prioritize which vulnerabilities to fix first, and ensure that they take care of critical vulnerabilities right away. You could use the following scale to communicate severity:

Low severity

The bug doesn't have the potential to cause a lot of damage. For example, an open redirect that can be used only for phishing is a low-severity bug.

Medium severity

The bug impacts users or the organization in a moderate way, or is a high-severity issue that's difficult for a malicious hacker to exploit. The security team should focus on high- and critical-severity bugs first. For example, a cross-site request forgery (CSRF) on a sensitive action such as password change is often considered a medium-severity issue.

High severity

The bug impacts a large number of users, and its consequences can be disastrous for these users. The security team should fix a high-security bug as soon as possible. For example, an open redirect that can be used to steal OAuth tokens is a high-severity bug.

Critical severity

The bug impacts a majority of the user base or endangers the organization's core infrastructure. The security team should fix a critical-severity bug right away. For example, a SQL injection leading to remote code execution (RCE) on the production server will be considered a critical issue.

Study the Common Vulnerability Scoring System (CVSS) at *https://www.first.org/cvss/* for a general idea of how critical each type of vulnerability is. The CVSS scale takes into account factors such as how a vulnerability impacts an organization, how hard the vulnerability is to exploit, and whether the vulnerability requires any special privileges or user interaction to exploit.

Then, try to imagine what your client company cares about, and which vulnerabilities would present the biggest business impact. Customize your assessment to fit into the client's business priorities. For example, a dating site might find a bug that exposes a user's birth date as inconsequential, since a user's age is already public information on the site, while a job search site might find a similar bug significant, because an applicant's age should be confidential in the job search process. On the other hand, leaks of users' banking information are almost always considered a high-severity issue.

If you're unsure which severity rating your bug falls into, use the rating scale of a bug bounty platform. For example, Bugcrowd's rating system takes into account the type of vulnerability and the affected functionality (*https://bugcrowd.com/vulnerability-rating-taxonomy/*), and HackerOne provides a severity calculator based on the CVSS scale (*https://docs.hackerone.com/hackers/severity.html*).

You could list the severity in a single line, as follows:

```
Severity of the issue: High
```

Providing an accurate assessment of severity will make everyone's lives easier and contribute to a positive relationship between you and the security team.

Step 4: Give Clear Steps to Reproduce

Next, provide step-by-step instructions for reproducing the vulnerability. Include all relevant setup prerequisites and details you can think of. It's best to assume the engineer on the other side has no knowledge of the vulnerability and doesn't know how the application works.

For example, a merely okay report might include the following steps to reproduce:

1. Log onto the site and visit *https://example.com/change_password*.
2. Click the **Change Password** button.
3. Intercept the request, and change the `user_id` parameter to another user's ID.

Notice that these steps aren't comprehensive or explicit. They don't specify that you need two test accounts to test for the vulnerability. They also assume that you have enough knowledge about the application and the format of its requests to carry out each step without more instructions.

Now, here is another example in a better report:

Make two accounts on *example.com*: A and user B.

1. Log onto *example.com* as account A, and visit *https://example.com/change_password*.
2. Fill in the desired new password in the **New password** field, located at the top left of the page.
3. Click the **Change Password** button located at the top right of the page.
4. Intercept the POST request to *https://example.com/change_password* and change the `user_id` POST parameter to the user ID of account B.
5. You can now log into account B by using the new password you've chosen.

Although the security team will probably still understand the first report, the second report is a lot more specific. By providing many relevant details, you can avoid any misunderstanding and speed up the mitigation process.

Step 5: Provide a Proof of Concept

For simple vulnerabilities, the steps you provide might be all that the security team needs to reproduce the issue. But for more-complex vulnerabilities, it's helpful to include a video, screenshots, or photos documenting your exploit, called a *proof-of-concept (POC)* file.

For example, for a CSRF vulnerability, you could include an HTML file with the CSRF payload embedded. This way, all the security team needs to do to reproduce the issue is to open the HTML file in their browser. For an XML external entity attack, include the crafted XML file that you used to execute the attack. And for vulnerabilities that require multiple complicated steps to reproduce, you could film a screen-capture video of you walking through the process.

POC files like these save the security team time because they won't have to prepare the attack payload themselves. You can also include any crafted URLs, scripts, or upload files you used to attack the application.

Step 6: Describe the Impact and Attack Scenarios

To help the security team fully understand the potential impact of the vulnerability, you can also illustrate a plausible scenario in which the vulnerability could be exploited. Note that this section is not the same as the severity assessment I mentioned earlier. The severity assessment describes the severity of the consequences of an attacker exploiting the vulnerability, whereas the attack scenario explains what those consequences would actually look like.

If hackers exploited this bug, could they take over user accounts? Or could they steal user information and cause large-scale data leaks? Put yourself in a malicious hacker's shoes and try to escalate the impact of the vulnerability as much as possible. Give the client company a realistic sense of the worst-case scenario. This will help the company prioritize the fix internally and determine if any additional steps or internal investigations are necessary.

Here is an example of an impact section:

> **Impact:**
>
> Using this vulnerability, all that an attacker needs in order to change a user's password is their `user_id`. Since each user's public profile page lists the account's `user_id`, anyone can visit any user's profile, find out their `user_id`, and change their password. And because `user_id`s are simply sequential numbers, a hacker can even enumerate all the `user_id`s and change the passwords of all users! This bug will let attackers take over anyone's account with minimal effort.

A good impact section illustrates how an attacker can realistically exploit a bug. It takes into account any mitigating factors as well as the maximum impact that can be achieved. It should never overstate a bug's impact or include any hypotheticals.

Step 7: Recommend Possible Mitigations

You can also recommend possible steps the security team can take to mitigate the vulnerability. This will save the team time when it begins researching mitigations. Often, since you're the security researcher who discovered the vulnerability, you'll be familiar with the particular behavior of that application feature, and thus in a good position to come up with a comprehensive fix.

However, don't propose fixes unless you have a good understanding of the root cause of the issue. Internal teams may have much more context and expertise to provide appropriate mitigation strategies applicable to

their environment. If you're not sure what caused the vulnerability or what a possible fix might be, avoid giving any recommendations so you don't confuse your reader.

Here is a possible mitigation you could propose:

> The application should validate the user's `user_id` parameter within the change password request to ensure that the user is authorized to make account modifications. Unauthorized requests should be rejected and logged by the application.

You don't have to go into the technical details of the fix, since you don't have knowledge of the application's underlying codebase. But as someone who understands the vulnerability class, you can provide a direction for mitigation.

Step 8: Validate the Report

Finally, always validate your report. Go through your report one last time to make sure that there are no technical errors, or anything that might prevent the security team from understanding it. Follow your own steps to reproduce to ensure that they contain enough details. Examine all of your POC files and code to make sure they work. By validating your reports, you can minimize the possibility of submitting an invalid report.

Additional Tips for Writing Better Reports

Here are additional tips to help you deliver the best reports possible.

Don't Assume Anything

First, don't assume that the security team will be able to understand everything in your report. Remember that you might have been working with this vulnerability for a week, but to the security team receiving the report, it's all new information. They have a whole host of other responsibilities on their plates and often aren't as familiar with the feature as you. Additionally, reports are not always assigned to security teams. Newer programs, open source projects, and startups may depend on developers or technical support personnel to handle bug reports instead of having a dedicated security team. Help them understand what you've discovered.

Be as verbose as possible, and include all the relevant details you can think of. It's also good to include links to references explaining obscure security knowledge that the security team might not be familiar with. Think about the potential consequences of being verbose versus the consequences of leaving out essential details. The worst thing that can happen if you're too wordy is that your report will take two extra minutes to read. But if you leave out important details, the remediation of the vulnerability might get delayed, and a malicious hacker might exploit the bug.

Be Clear and Concise

On the other hand, don't include any unnecessary information, such as wordy greetings, jokes, or memes. A security report is a business document,

not a letter to your friend. It should be straightforward and to the point. Make your report as short as possible without omitting the key details. You should always be trying to save the security team's time so they can get to remediating the vulnerability right away.

Write What You Want to Read

Always put your reader in mind when writing, and try to build a good reading experience for them. Write in a conversational tone and don't use leetspeak, slang, or abbreviations. These make the text harder to read and will add to your reader's annoyance.

Be Professional

Finally, always communicate with the security team with respect and professionalism. Provide clarifications regarding the report patiently and promptly.

You'll probably make mistakes when writing reports, and miscommunication will inevitably happen. But remember that as the security researcher, you have the power to minimize that possibility by putting time and care into your writing. By honing your reporting skills in addition to your hacking skills, you can save everyone's time and maximize your value as a hacker.

Building a Relationship with the Development Team

Your job as a hacker doesn't stop the moment you submit the report. As the person who discovered the vulnerability, you should help the company fix the issue and make sure the vulnerability is fully patched.

Let's talk about how to handle your interactions with the security team after the report submission, and how to build strong relationships with them. Building a strong relationship with the security team will help get your reports resolved more quickly and smoothly. It might even lead to bigger bug bounty payouts if you can consistently contribute to the security of the organization. Some bug bounty hunters have even gotten interviews or job offers from top tech firms because of their bug bounty findings! We'll go over the different states of your report, what you should do during each stage of the mitigation process, and how to handle conflicts when communicating with the security team.

Understanding Report States

Once you've submitted your report, the security team will classify it into a *report state*, which describes the current status of your report. The report state will change as the process of mitigation moves forward. You can find the report state listed on the bug bounty platform's interface, or in the messages you receive from security teams.

Need More Information

One of the most common report states you'll see is *need more information*. This means the security team didn't fully understand your report, or couldn't reproduce the issue by using the information you've provided. The security team will usually follow up with questions or requests for additional information about the vulnerability.

In this case, you should revise your report, provide any missing information, and address the security team's additional concerns.

Informative

If the security team marks your report as *informative*, they won't fix the bug. This means they believe the issue you reported is a security concern but not significant enough to warrant a fix. Vulnerabilities that do not impact other users, such as the ability to increase your own scores on an online game, often fall into this category. Another type of bug often marked as informative is a missing security best practice, like allowing users to reuse passwords.

In this case, there's nothing more you can do for the report! The company won't pay you a bounty, and you don't have to follow up, unless you believe the security team made a mistake. However, I do recommend that you keep track of informative issues and try to chain them into bigger, more impactful bugs.

Duplicate

A *duplicate* report status means another hacker has already found the bug, and the company is in the process of remediating the vulnerability.

Unfortunately, since companies award bug bounties to only the first hacker who finds the bug, you won't get paid for duplicates. There's nothing more to do with the report besides helping the company resolve the issue. You can also try to escalate or chain the bug into a more impactful bug. That way, the security team might see the new report as a separate issue and reward you.

N/A

A *not applicable (N/A)* status means your report doesn't contain a valid security issue with security implications. This might happen when your report contains technical errors, or if the bug is intentional application behavior.

N/A reports don't pay. There is nothing more for you to do here besides move on and continue hacking!

Triaged

Security teams *triage* a report when they've validated the report on their end. This is great news for you, because this usually means the security team is going to fix the bug and reward you with a bounty.

Once the report has been triaged, you should help the security team fix the issue. Follow up with their questions promptly, and provide any additional information they ask for.

Resolved

When your report is marked as *resolved*, the reported vulnerability has been fixed. At this point, pat yourself on the back and rejoice in the fact that you've made the internet a little safer. If you are participating in a paid bug bounty program, you can also expect to receive your payment at this point!

There's nothing more to do with the report besides celebrate and continue hacking.

Dealing with Conflict

Not all reports can be resolved quickly and smoothly. Conflicts inevitably happen when the hacker and the security team disagree on the validity of the bug, the severity of the bug, or the appropriate payout amount. Even so, conflicts could ruin your reputation as a hacker, so handling them professionally is key to a successful bug hunting career. Here's what you should do if you find yourself in conflict with the security team.

When you disagree with the security team about the validity of the bug, first make sure that all the information in your initial report is correct. Often, security teams mark reports as informative or N/A because of a technical or writing mistake. For example, if you included incorrect URLs in your POC, the security team might not be able to reproduce the issue. If this caused the disagreement, send over a follow-up report with the correct information as soon as possible.

On the other hand, if you didn't make a mistake in your report but still believe they've labeled the issue incorrectly, send a follow-up explaining why you believe that the bug is a security issue. If that still doesn't resolve the misunderstanding, you can ask for mediation by the bug bounty platform or other security engineers on the team.

Most of the time, it is difficult for others to see the impact of a vulnerability if it doesn't belong to a well-known bug class. If the security team dismisses the severity of the reported issue, you should explain some potential attack scenarios to fully illustrate its impact.

Finally, if you're unhappy with the bounty amount, communicate that without resentment. Ask for the organization's reasoning behind assigning that bounty, and explain why you think you deserve a higher reward. For example, if the person in charge of your report underestimated the severity of the bug, you can elaborate on the impact of the issue when you ask for a higher reward. Whatever you do, always avoid asking for more money without explanation.

Remember, we all make mistakes. If you believe the person handling your report mishandled the issue, ask for reconsideration courteously. Once you've made your case, respect the company's final decision about the fix and bounty amount.

Building a Partnership

The bug bounty journey doesn't stop after you've resolved a report. You should strive to form long-term partnerships with organizations. This can

help get your reports resolved more smoothly and might even land you an interview or job offer. You can form good relationships with companies by respecting their time and communicating with professionalism.

First, gain respect by always submitting validated reports. Don't break a company's trust by spamming, pestering them for money, or verbally abusing the security team. In turn, they'll respect you and prioritize you as a researcher. Companies often ban hunters who are disrespectful or unreasonable, so avoid falling into those categories at all costs.

Also learn the communication style of each organization you work with. How much detail do they expect in their reports? You can learn about a security team's communication style by reading their publicly disclosed reports, or by incorporating their feedback about your reports into future messages. Do they expect lots of photos and videos to document the bug? Customize your reports to make your reader's job easier.

Finally, make sure you support the security team until they resolve the issue. Many organizations will pay you a bounty upon report triage, but please don't bail on the security team after you receive the reward! If it's requested, provide advice to help mitigate the vulnerability, and help security teams confirm that the issue has been fixed. Sometimes organizations will ask you to perform retests for a fee. Always take that opportunity if you can. You'll not only make money, but also help companies resolve the issue faster.

Understanding Why You're Failing

You've poured hours into looking for vulnerabilities and haven't found a single one. Or you keep submitting reports that get marked informative, N/A, or duplicate.

You've followed all the rules. You've used all the tools. What's going wrong? What secrets are the leaderboard hackers hiding from you? In this section, I'll discuss the mistakes that prevent you from succeeding in bug bounties, and how you can improve.

Why You're Not Finding Bugs

If you spend a lot of time in bug bounties and still have trouble finding bugs, here are some possible reasons.

You Participate in the Wrong Programs

You might have been targeting the wrong programs all along. Bug bounty programs aren't created equally, and picking the right one is essential. Some programs delay fixing bugs because they lack the resources to deal with reports. Some programs downplay the severity of vulnerabilities to avoid paying hackers. Finally, other programs restrict their scope to a small subset of their assets. They run bug bounty programs to gain positive publicity and don't intend to actually fix vulnerabilities. Avoid these programs to save yourself the headache.

You can identify these programs by reading publicly disclosed reports, analyzing program statistics on bug bounty platforms, or by discussing with other hackers. A program's stats listed on bug bounty platforms provide a lot of information on how well a program is executed. Avoid programs with long response times and programs with low average bounties. Pick targets carefully, and prioritize companies that invest in their bug bounty programs.

You Don't Stick to a Program

How long should you target a program? If your answer is a few hours or days, that's the reason you're not finding anything. Jumping from program to program is another mistake beginners often make.

Every bug bounty program has countless bug bounty hunters hacking it. Differentiate yourself from the competition, or risk not finding anything! You can differentiate yourself in two ways: dig deep or search wide. For example, dig deep into a single functionality of an application to search for complex bugs. Or discover and hack the lesser-known assets of the company.

Doing these things well takes time. Don't expect to find bugs right away when you're starting fresh on a program. And don't quit a program if you can't find bugs on the first day.

You Don't Recon

Jumping into big public programs without performing reconnaissance is another way to fail at bug bounties. Effective recon, which we discuss in Chapter 5, helps you discover new attack surfaces: new subdomains, new endpoints, and new functionality.

Spending time on recon gives you an incredible advantage over other hackers, because you'll be the first to notice the bugs on all assets you discover, giving you better chances of finding bugs that aren't duplicates.

You Go for Only Low-Hanging Fruit

Another mistake that beginners often make is to rely on vulnerability scanners. Companies routinely scan and audit their applications, and other bug bounty hunters often do the same, so this approach won't give you good results.

Also, avoid looking for only the obvious bug types. Simplistic bugs on big targets have probably already been found. Many bug bounty programs were private before companies opened them to the public. This means a few experienced hackers will have already reported the easiest-to-find bugs. For example, many hackers will likely have already tested for a stored-XSS vulnerability on a forum's comment field.

This isn't to say that you shouldn't look for low-hanging fruit at all. Just don't get discouraged if you don't find anything that way. Instead, strive to gain a deeper understanding of the application's underlying architecture and logic. From there, you can develop a unique testing methodology that will result in more unique and valuable bugs.

You Don't Get into Private Programs

It becomes much easier to find bugs after you start hacking on private programs. Many successful hackers say that most of their findings come from private programs. Private programs are a lot less crowded than public ones, so you'll have less competition, and less competition usually means more easy finds and fewer duplicates.

Why Your Reports Get Dismissed

As mentioned, three types of reports won't result in a bounty: N/As, informatives, and duplicates. In this section, I'll talk about what you can do to reduce these disappointments.

Reducing the number of invalid reports benefits everyone. It will not only save you time and effort, but also save the security team the staff hours dedicated to processing these reports. Here are some reasons your reports keep getting dismissed.

You Don't Read the Bounty Policy

One of the most common reasons reports get marked as N/A is that they're out of scope. A program's policy page often has a section labeled *Scope* that tells you which of the company's assets you're allowed to hack. Most of the time, the policy page also lists vulnerabilities and assets that are *out of scope*, meaning you're not allowed to report about them.

The best way to prevent submitting N/As is to read the bounty policy carefully, and repeatedly. Which vulnerability types are out of scope? And which of the organization's assets? Respect these boundaries, and don't submit bugs that are out of scope.

If you do accidentally find a critical issue that is out of scope, report it if you think it's something that the organization has to know about! You might not get rewarded, but you can still contribute to the company's security.

You Don't Put Yourself in the Organization's Shoes

Informative reports are much harder to prevent than N/As. Most of the time, you'll get informative ratings because the company doesn't care about the issue you're reporting.

Imagine yourself as a security engineer. If you're busy safeguarding millions of users' data every day, would you care about an open redirect that can be used only for phishing? Although it's a valid security flaw, you probably wouldn't. You have other responsibilities to tend to, so fixing a low-severity bug is at the bottom of your to-do list. If the security team does not have the extra staff to deal with these reports, they will sometimes ignore it and mark it as informative.

I've found that the most helpful way to reduce informatives is to put myself in the organization's shoes. Learn about the organization so you can identify its product, the data it's protecting, and the parts of its application that are the most important. Once you know the business's priorities, you can go after the vulnerabilities that the security team cares about.

And remember, different companies have different priorities. An informative report to one organization could be a critical one to another. Like the dating site versus job search site example mentioned earlier in this chapter, everything is relative. Sometimes, it's difficult to figure out how important a bug will be to an organization. Some issues I've reported as critical ended up being informative. And some vulnerabilities I classified as low impact were rewarded as critical issues.

This is where trial and error can pay off. Every time the security team classifies your report as informative, take note for future reference. The next time you find a bug, ask yourself: did this company care about issues like this in the past? Learn what each company cares about, and tailor your hacking efforts to suit their business priorities. You'll eventually develop an intuition about what kinds of bugs deliver the most impact.

You Don't Chain Bugs

You might also be getting informatives because you always report the first minor bug you find.

But minor bugs classified as informative can become big issues if you learn to chain them. When you find a low-severity bug that might get dismissed, don't report it immediately. Try to use it in future bug chains instead. For example, instead of reporting an open redirect, use it in a server-side request forgery (SSRF) attack!

You Write Bad Reports

Another mistake beginners often make is that they fail to communicate the bug's impact in their report. Even when a vulnerability is impactful, if you can't communicate its implications to the security team, they'll dismiss the report.

What About Duplicates?

Unfortunately, sometimes you can't avoid duplicates. But you could lower your chances of getting duplicates by hunting on programs with large scopes, hacking on private programs, performing recon extensively, and developing your unique hunting methodology.

What to Do When You're Stuck

When I got started in bug bounties, I often went days or weeks without finding a single vulnerability. My first ever target was a social media site with a big scope. But after reporting my first CSRFs and IDORs, I soon ran out of ideas (and luck). I started checking for the same vulnerabilities over and over again, and trying out different automatic tools, to no avail.

I later found out I wasn't alone; this type of *bug slump* is surprisingly common among new hackers. Let's talk about how you can bounce back from frustration and improve your results when you get stuck.

Step 1: Take a Break!

First, take a break. Hacking is hard work. Unlike what they show in the movies, hunting for vulnerabilities is tedious and difficult. It requires patience, persistence, and an eye for detail, so it can be very mentally draining.

Before you keep hacking away, ask yourself: am I tired? A lack of inspiration could be your brain's way of telling you it has reached its limits. In this case, your best course of action would be to rest it out. Go outside. Meet up with friends. Have some ice cream. Or stay inside. Make some tea. And read a good book.

There is more to life than SQL injections and XSS payloads. If you take a break from hacking, you'll often find that you're much more creative when you come back.

Step 2: Build Your Skill Set

Use your hacking slump as an opportunity to improve your skills. Hackers often get stuck because they get too comfortable with certain familiar techniques, and when those techniques don't work anymore, they mistakenly assume there's nothing left to try. Learning new skills will get you out of your comfort zone and strengthen your hacker skills for the future.

First, if you're not already familiar with the basic hacking techniques, refer to testing guides and best practices to solidify your skills. For example, the Open Web Application Security Project (OWASP) has published testing guides for various asset types. You can find OWASP's web and mobile testing guides at *https://owasp.org/www-project-web-security-testing-guide/* and *https://owasp.org/www-project-mobile-security-testing-guide/*.

Learn a new hacking technique, whether it's a new web exploitation technique, a new recon angle, or a different platform, such as Android. Focus on a specific skill you want to build, read about it, and apply it to the targets you're hacking. Who knows? You might uncover a whole new way to approach the target application! You can also take this opportunity to catch up with what other hackers are doing by reading the many hacker blogs and write-up sites out there. Understanding other hackers' approaches can provide you with a refreshing new perspective on engaging with your target.

Next, play *Capture the Flags (CTFs)*. In these security competitions, players search for flags that prove that they've hacked into a system. CTFs are a great way to learn about new vulnerabilities. They're also fun and often feature interesting new classes of vulnerabilities. Researchers are constantly discovering new kinds of vulnerabilities, and staying on top of these techniques will ensure that you're constantly finding bugs.

Step 3: Gain a Fresh Perspective

When you're ready to hack live targets again, here are some tips to help you keep your momentum.

First, hacking on a single target can get boring, so diversify your targets instead of focusing on only one. I've always found it helpful to have a few targets to alternate between. When you're getting tired of one application, switch to another, and come back to the first one later.

Second, make sure you're looking for specific things in a target instead of wandering aimlessly, searching for anything. Make a list of the new skills you've learned and try them out. Look for a new kind of bug, or try out a new recon angle. Then, rinse and repeat until you find a suitable new workflow.

Finally, remember that hacking is not always about finding a single vulnerability but combining several weaknesses of an application into something critical. In this case, it's helpful to specifically look for weird behavior instead of vulnerabilities. Then take note of these weird behaviors and weaknesses, and see if you can chain them into something worth reporting.

Lastly, a Few Words of Experience

Bug bounty hunting is difficult. When I started hunting for bugs, I'd sometimes go months without finding one. And when I did find one, it'd be something trivial and low severity.

The key to getting better at anything is practice. If you're willing to put in the time and effort, your hacking skills will improve, and you'll soon see yourself on leaderboards and private invite lists! If you get frustrated during this process, remember that everything gets easier over time. Reach out to the hacker community if you need help. And good luck!

II

GETTING STARTED

3

HOW THE INTERNET WORKS

Before you jump into hunting for bugs, let's take some time to understand how the internet works. Finding web vulnerabilities is all about exploiting weaknesses in this technology, so all good hackers should have a solid understanding of it. If you're already familiar with these processes, feel free to skip ahead to my discussion of the internet's security controls.

The following question provides a good starting place: what happens when you enter *www.google.com* in your browser? In other words, how does your browser know how to go from a domain name, like google.com, to the web page you're looking for? Let's find out.

The Client-Server Model

The internet is composed of two kind of devices: *clients* and *servers*. *Clients* request resources or services, and *servers* provide those resources and services. When you visit a website with your browser, it acts as a client and requests a web page from a web server. The web server will then send your browser the web page (Figure 3-1).

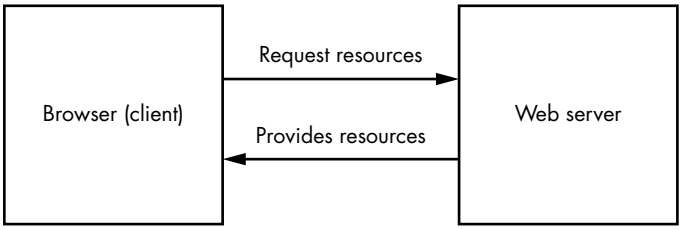


Figure 3-1: Internet clients request resources from servers.

A web page is nothing more than a collection of resources or files sent by the web server. For example, at the very least, the server will send your browser a text file written in *Hypertext Markup Language (HTML)*, the language that tells your browser what to display. Most web pages also include Cascading Style Sheets (CSS) files to make them pretty. Sometimes web pages also contain JavaScript (JS) files, which enable sites to animate the web page and react to user input without going through the server. For example, JavaScript can resize images as users scroll through the page and validate a user input on the client side before sending it to the server. Finally, your browser might receive embedded resources, such as images and videos. Your browser will combine these resources to display the web page you see.

Servers don't just return web pages to the user, either. Web APIs enable applications to request the data of other systems. This enables applications to interact with each other and share data and resources in a controlled way. For example, Twitter's APIs allow other websites to send requests to Twitter's servers to retrieve data such as lists of public Tweets and their authors. APIs power many internet functionalities beyond this, and we'll revisit them, along with their security issues, in Chapter 24.

The Domain Name System

How do your browser and other web clients know where to find these resources? Well, every device connected to the internet has a unique Internet Protocol (IP) address that other devices can use to find it. However, IP addresses are made up of numbers and letters that are hard for humans to remember. For example, the older format of IP addresses, IPv4, looks like this: 123.45.67.89. The new version, IPv6, looks even more complicated: 2001:db8::ff00:42:8329.

This is where the *Domain Name System (DNS)* comes in. A DNS server functions as the phone book for the internet, translating domain names into IP addresses (Figure 3-2). When you enter a domain name in your browser, a DNS server must first convert the domain name into an IP address. Our browser asks the DNS server, "Which IP address is this domain located at?"

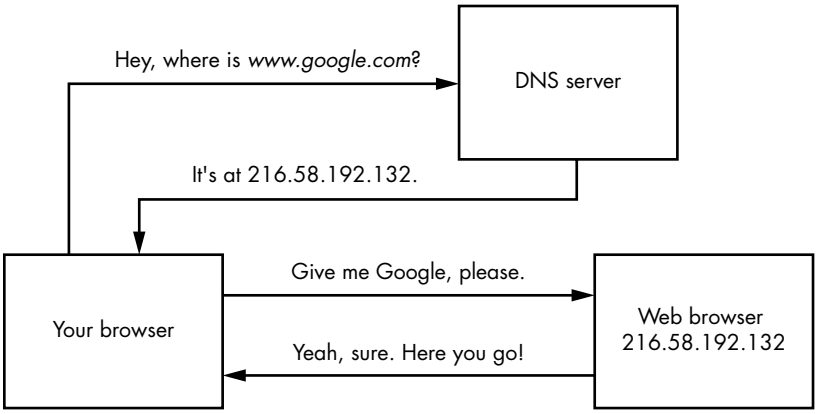


Figure 3-2: A DNS server will translate a domain name to an IP address.

Internet Ports

After your browser acquires the correct IP address, it will attempt to connect to that IP address via a port. A *port* is a logical division on devices that identifies a specific network service. We identify ports by their port numbers, which can range from 0 to 65,535.

Ports allow a server to provide multiple services to the internet at the same time. Because conventions exist for the traffic received on certain ports, port numbers also allow the server to quickly forward arriving internet messages to a corresponding service for processing. For example, if an internet client connects to port 80, the web server understands that the client wishes to access its web services (Figure 3-3).

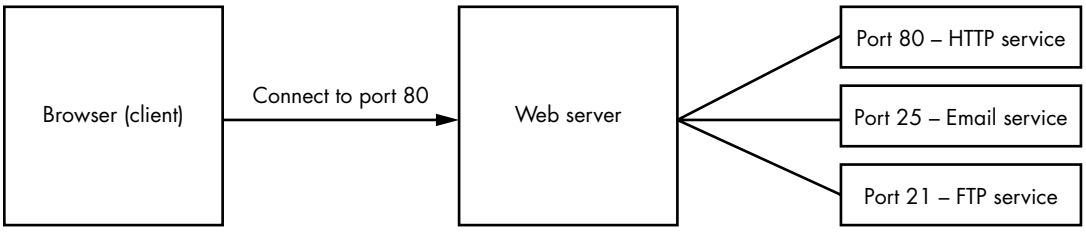


Figure 3-3: Ports allow servers to provide multiple services. And port numbers help forward client requests to the right service.

By default, we use port 80 for HTTP messages and port 443 for HTTPS, the encrypted version of HTTP.

HTTP Requests and Responses

Once a connection is established, the browser and server communicate via the *HyperText Transfer Protocol (HTTP)*. HTTP is a set of rules that specifies how to structure and interpret internet messages, and how web clients and web servers should exchange information.

When your browser wants to interact with a server, it sends the server an *HTTP request*. There are different types of HTTP requests, and the two most common are GET and POST. By convention, GET requests retrieve data from the server, while POST requests submit data to it. Other common HTTP methods include OPTIONS, used to request permitted HTTP methods for a given URL; PUT, used to update a resource; and DELETE, used to delete a resource.

Here is an example GET request that asks the server for the home page of *www.google.com*:

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-US
Accept-Encoding: gzip, deflate
Connection: close
```

Let's walk through the structure of this request, since you'll be seeing a lot of these in this book. All HTTP requests are composed of a request line, request headers, and an optional request body. The preceding example contains only the request line and headers.

The *request line* is the first line of the HTTP request. It specifies the request method, the requested URL, and the version of HTTP used. Here, you can see that the client is sending an HTTP GET request to the home page of *www.google.com* using HTTP version 1.1.

The rest of the lines are HTTP *request headers*. These are used to pass additional information about the request to the server. This allows the server to customize results sent to the client. In the preceding example, the `Host` header specifies the hostname of the request. The `User-Agent` header contains the operating system and software version of the requesting software, such as the user's web browser. The `Accept`, `Accept-Language`, and `Accept-Encoding` headers tell the server which format the responses should be in. And the `Connection` header tells the server whether the network connection should stay open after the server responds.

You might see a few other common headers in requests. The `Cookie` header is used to send cookies from the client to the server. The `Referer` header specifies the address of the previous web page that linked to the current page. And the `Authorization` header contains credentials to authenticate a user to a server.

After the server receives the request, it will try to fulfill it. The server will return all the resources used to construct your web page by using *HTTP responses*. An HTTP response contains multiple things: an HTTP status code to indicate whether the request succeeded; HTTP headers, which are

bits of information that browsers and servers use to communicate with each other about authentication, content format, and security policies; and the HTTP response body, or the actual web content that you requested. The web content could include HTML code, CSS style sheets, JavaScript code, images, and more.

Here is an example of an HTTP response:

```
❶ HTTP/1.1 200 OK
❷ Date: Tue, 31 Aug 2021 17:38:14 GMT
  […]
❸ Content-Type: text/html; charset=UTF-8
❹ Server: gws
❺ Content-Length: 190532

  <!doctype html>
  […]
  <title>Google</title>
  […]
  <html>
```

Notice the `200 OK` message on the first line ❶. This is the status code. An HTTP status code in the 200 range indicates a successful request. A status code in the 300 range indicates a redirect to another page, whereas the 400 range indicates an error on the client's part, like a request for a nonexistent page. The 500 range means that the server itself ran into an error.

As a bug bounty hunter, you should always keep an eye on these status codes, because they can tell you a lot about how the server is operating. For example, a status code of 403 means that the resource is forbidden to you. This might mean that sensitive data is hidden on the page that you could reach if you can bypass the access controls.

The next few lines separated by a colon (`:`) in the response are the HTTP response headers. They allow the server to pass additional information about the response to the client. In this case, you can see that the time of the response was `Tue, 31 Aug 2021 17:38:14 GMT` ❷. The `Content-Type` header indicates the file type of the response body. In this case, The `Content-Type` of this page is `text/html` ❸. The server version is Google Web Server (`gws`) ❹, and the `Content-Length` is 190,532 bytes ❺. Usually, additional response headers will specify the content's format, language, and security policies.

In addition to these, you might encounter a few other common response headers. The `Set-Cookie` header is sent by the server to the client to set a cookie. The `Location` header indicates the URL to which to redirect the page. The `Access-Control-Allow-Origin` header indicates which origins can access the page's content. (We will talk about this more in Chapter 19.) `Content-Security-Policy` controls the origin of the resources the browser is allowed to load, while the `X-Frame-Options` header indicates whether the page can be loaded within an iframe (discussed further in Chapter 8).

The data after the blank line is the response body. It contains the actual content of the web page, such as the HTML and JavaScript code. Once your browser receives all the information needed to construct the web page, it will render everything for you.

Internet Security Controls

Now that you have a high-level understanding of how information is communicated over the internet, let's dive into some fundamental security controls that protect it from attackers. To hunt for bugs effectively, you will often need to come up with creative ways to bypass these controls, so you'll first need to understand how they work.

Content Encoding

Data transferred in HTTP requests and responses isn't always transmitted in the form of plain old text. Websites often encode their messages in different ways to prevent data corruption.

Data encoding is used as a way to transfer binary data reliably across machines that have limited support for different content types. Characters used for encoding are common characters not used as controlled characters in internet protocols. So when you encode content using common encoding schemes, you can be confident that your data is going to arrive at its destination uncorrupted. In contrast, when you transfer your data in its original state, the data might be screwed up when internet protocols misinterpret special characters in the message.

Base64 encoding is one of the most common ways of encoding data. It's often used to transport images and encrypted information within web messages. This is the base64-encoded version of the string `"Content Encoding"`:

```
Q29udGVudCBFbmNvZGluZw==
```

Base64 encoding's character set includes the uppercase alphabet characters A to Z, the lowercase alphabet characters a to z, the number characters 0 to 9, the characters + and /, and finally, the = character for padding. *Base64url encoding* is a modified version of base64 used for the URL format. It's similar to base64, but uses different non-alphanumeric characters and omits padding.

Another popular encoding method is hex encoding. *Hexadecimal encoding*, or *hex*, is a way of representing characters in a base-16 format, where characters range from 0 to F. Hex encoding takes up more space and is less efficient than base64 but provides for a more human-readable encoded string. This is the hex-encoded version of the string `"Content Encoding"`; you can see that it takes up more characters than its base64 counterpart:

```
436f6e74656e7420456e636f64696e67
```

URL encoding is a way of converting characters into a format that is more easily transmitted over the internet. Each character in a URL-encoded string can be represented by its designated hex number preceded by a % symbol. See Wikipedia for more information about URL encoding: *https://en.wikipedia.org/wiki/Percent-encoding*.

For example, the word *localhost* can be represented with its URL-encoded equivalent, `%6c%6f%63%61%6c%68%6f%73%74`. You can calculate a hostname's URL-encoded equivalent by using a URL calculator like URL Decode and Encode (*https://www.urlencoder.org/*).

We'll cover a couple of additional types of character encoding—octal encoding and dword encoding—when we discuss SSRFs in Chapter 13. When you see encoded content while investigating a site, always try to decode it to discover what the website is trying to communicate. You can use Burp Suite's decoder to decode encoded content. We'll cover how to do this in the next chapter. Alternatively, you can use CyberChef (*https://gchq.github.io/CyberChef/*) to decode both base64 content and other types of encoded content.

Servers sometimes also encrypt their content before transmission. This keeps the data private between the client and server and prevents anyone who intercepts the traffic from eavesdropping on the messages.

Session Management and HTTP Cookies

Why is it that you don't have to re-log in every time you close your email tab? It's because the website remembers your session. *Session management* is an organizational process that allows the server to handle multiple requests from the same user without asking the user to log in again.

Websites maintain a session for each logged-in user, and a new session starts when you log into the website (Figure 3-4). The server will assign an associated *session ID* for your browser that serves as proof of your identity. The session ID is usually a long and unpredictable sequence designed to be unguessable. When you log out, the server ends the session and revokes the session ID. The website might also end sessions periodically if you don't manually log out.

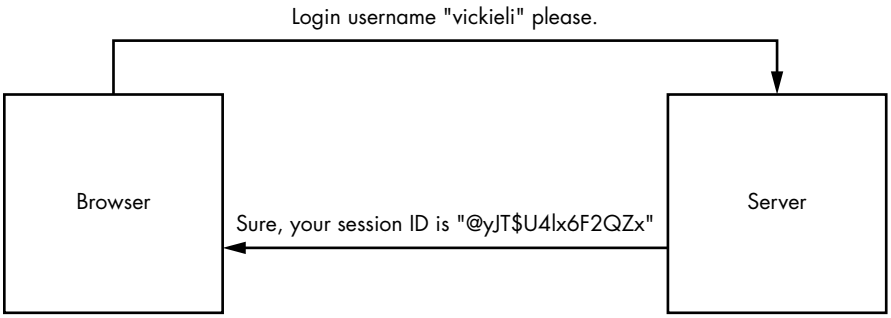


Figure 3-4: After you log in, the server creates a session for you and issues a session ID, which uniquely identifies a session.

Most websites use cookies to communicate session information in HTTP requests. *HTTP cookies* are small pieces of data that web servers send to your browser. When you log in to a site, the server creates a session for you and sends the session ID to your browser as a cookie. After receiving a cookie, your browser stores it and includes it in every request to the same server. (Figure 3-5).

That's how the server knows it's you! After the cookie for the session is generated, the server will track it and use it to validate your identity. Finally, when you log out, the server will invalidate the session cookie so that it cannot be used again. The next time you log in, the server will create a new session and a new associated session cookie for you.

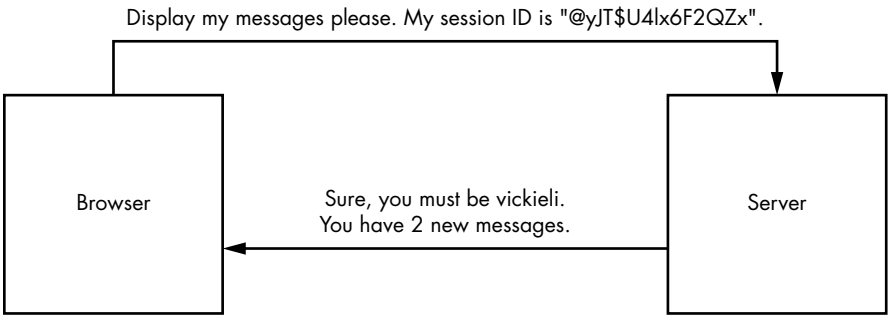


Figure 3-5: Your session ID correlates with session information that is stored on the server.

Token-Based Authentication

In session-based authentication, the server stores your information and uses a corresponding session ID to validate your identity, whereas a *token-based authentication* system stores this info directly in some sort of token. Instead of storing your information server-side and querying it using a session ID, tokens allow servers to deduce your identity by decoding the token itself. This way, applications won't have to store and maintain session information server-side.

This system comes with a risk: if the server uses information contained in the token to determine the user's identity, couldn't users modify the information in the tokens and log in as someone else? To prevent token forgery attacks like these, some applications encrypt their tokens, or encode the token so that it can be read by only the application itself or other authorized parties. If the user can't understand the contents of the token, they probably can't tamper with it effectively either. Encrypting a token does not prevent token forgery completely. There are ways that an attacker can tamper with an encrypted token without understanding its contents. But it's a lot more difficult than tampering with a plaintext token.

Another more reliable way applications protect the integrity of a token is by signing the token, and verifying the token signature when it arrives at the server. *Signatures* are used to verify the integrity of a piece of data. They are special strings that can be generated only if you know a secret key. Since there is no way of generating a valid signature without the secret key, and only the server knows what the secret key is, a valid signature suggests that the token is probably not altered by the client or any third party. Although the implementations by applications can vary, token-based authentication works like this:

1. The user logs in with their credentials.
2. The server validates those credentials and provides the user with a signed token.

3. The user sends the token with every request to prove their identity.
4. Upon receiving and validating the token, the server reads the user's identity information from the token and responds with confidential data.

JSON Web Tokens

The *JSON Web Token (JWT)* is one of the most commonly used types of authentication tokens. It has three components: a header, a payload, and a signature.

The *header* identifies the algorithm used to generate the signature. It's a base64url-encoded string containing the algorithm name. Here's what a JWT header looks like:

```
eyBhbGcgOiBIUzI1NiwgdHlwIDogSldUIHOK
```

This string is the base64url-encoded version of this text:

```
{ "alg" : "HS256", "typ" : "JWT" }
```

The *payload* section contains information about the user's identity. This section, too, is base64url encoded before being used in the token. Here's an example of the payload section, which is the base64url-encoded string of { "*user_name*" : "*admin*", }:

```
eyB1c2VyX25hbWUgOiBhZG1pbiB9Cg
```

Finally, the *signature* section validates that the user hasn't tampered with the token. It's calculated by concatenating the header with the payload, then signing it with the algorithm specified in the header, and a secret key. Here's what a JWT signature looks like:

```
4Hb/6ibbViPOzq9SJflsNGPWSk6B8F6EqVrkNjpXh7M
```

For this specific token, the signature was generated by signing the string `eyBhbGcgOiBIUzI1NiwgdHlwIDogSldUIHOK.eyB1c2VyX25hbWUgOiBhZG1pbiB9Cg` with the HS256 algorithm using the secret key `key`. The complete token concatenates each section (the header, payload, and signature), separating them with a period (`.`):

```
eyBhbGcgOiBIUzI1NiwgdHlwIDogSldUIHOK.eyB1c2VyX25hbWUgOiBhZG1pbiB9Cg.4Hb/6ibbVi
POzq9SJflsNGPWSk6B8F6EqVrkNjpXh7M
```

When implemented correctly, JSON web tokens provide a secure way to identify the user. When the token arrives at the server, the server can verify that the token has not been tampered with by checking that the signature is correct. Then the server can deduce the user's identity by using the information contained in the payload section. And since the user does not have access to the secret key used to sign the token, they cannot alter the payload and sign the token themselves.

But if implemented incorrectly, there are ways that an attacker can bypass the security mechanism and forge arbitrary tokens.

Manipulating the alg Field

Sometimes applications fail to verify a token's signature after it arrives at the server. This allows an attacker to simply bypass the security mechanism by providing an invalid or blank signature.

One way that attackers can forge their own tokens is by tampering with the `alg` field of the token header, which lists the algorithm used to encode the signature. If the application does not restrict the algorithm type used in the JWT, an attacker can specify which algorithm to use, which could compromise the security of the token.

JWT supports a `none` option for the algorithm type. If the `alg` field is set to `none`, even tokens with empty signature sections would be considered valid. Consider, for example, the following token:

```
eyAiYWxnIiA6ICJOb25lIiwgInR5cCIgOiAiSldUIiB9Cg.eyB1c2VyX25hbWUgOiBhZG1pbiB9Cg.
```

This token is simply the base64url-encoded versions of these two blobs, with no signature present:

```
{ "alg" : "none", "typ" : "JWT" } { "user" : "admin" }
```

This feature was originally used for debugging purposes, but if not turned off in a production environment, it would allow attackers to forge any token they want and impersonate anyone on the site.

Another way attackers can exploit the `alg` field is by changing the type of algorithm used. The two most common types of signing algorithms used for JWTs are HMAC and RSA. HMAC requires the token to be signed with a key and then later verified with the same key. When using RSA, the token would first be created with a private key, then verified with the corresponding public key, which anyone can read. It is critical that the secret key for HMAC tokens and the private key for RSA tokens be kept a secret.

Now let's say that an application was originally designed to use RSA tokens. The tokens are signed with a private key A, which is kept a secret from the public. Then the tokens are verified with public key B, which is available to anyone. This is okay as long as the tokens are always treated as RSA tokens. Now if the attacker changes the `alg` field to HMAC, they might be able to create valid tokens by signing the forged tokens with the RSA public key, B. When the signing algorithm is switched to HMAC, the token is still verified with the RSA public key B, but this time, the token can be signed with the same public key too.

Brute-Forcing the Key

It could also be possible to guess, or *brute-force*, the key used to sign a JWT. The attacker has a lot of information to start with: the algorithm used to sign the token, the payload that was signed, and the resulting signature. If

the key used to sign the token is not complex enough, they might be able to brute-force it easily. If an attacker is not able to brute-force the key, they might try leaking the secret key instead. If another vulnerability, like a directory traversal, external entity attack (XXE), or SSRF exists that allows the attacker to read the file where the key value is stored, the attacker can steal the key and sign arbitrary tokens of their choosing. We'll talk about these vulnerabilities in later chapters.

Reading Sensitive Information

Since JSON web tokens are used for access control, they often contain information about the user. If the token is not encrypted, anyone can base64-decode the token and read the token's payload. If the token contains sensitive information, it might become a source of information leaks. A properly implemented signature section of the JSON web token provides data integrity, not confidentiality.

These are just a few examples of JWT security issues. For more examples of JWT vulnerabilities, use the search term *JWT security issues*. The security of any authentication mechanism depends not only on its design, but also its implementation. JWTs can be secure, but only if implemented properly.

The Same-Origin Policy

The *same-origin policy (SOP)* is a rule that restricts how a script from one origin can interact with the resources of a different origin. In one sentence, the SOP is this: a script from page A can access data from page B only if the pages are of the same origin. This rule protects modern web applications and prevents many common web vulnerabilities.

Two URLs are said to have the same origin if they share the same protocol, hostname, and port number. Let's look at some examples. Page A is at this URL:

https://medium.com/@vickieli

It uses HTTPS, which, remember, uses port 443 by default. Now look at the following pages to determine which has the same origin as page A, according to the SOP:

https://medium.com/

http://medium.com/

https://twitter.com/@vickieli

https://medium.com:8080/@vickieli

The *https://medium.com/* URL is of the same origin as page A, because the two pages share the same origin, protocol, hostname, and port number. The other three pages do not share the same origin as page A. *http://medium.com/* is of different origins from page A, because their protocols differ. *https://medium.com/* uses HTTPS, whereas *http://medium.com/* uses

HTTP. *https://twitter.com/@vickieli* is of a different origin as well, because it has a different hostname. Finally, *https://medium.com:8080/@vickieli* is of a different origin because it uses port 8080, instead of port 443.

Now let's consider an example to see how SOP protects us. Imagine that you're logged in into your banking site at *onlinebank.com*. Unfortunately, you click on a malicious site, *attacker.com*, in the same browser.

The malicious site issues a GET request to *onlinebank.com* to retrieve your personal information. Since you're logged into the bank, your browser automatically includes your cookies in every request you send to *onlinebank.com*, even if the request is generated by a script on a malicious site. Since the request contains a valid session ID, the server of *onlinebank.com* fulfills the request by sending the HTML page containing your info. The malicious script then reads and retrieves the private email addresses, home addresses, and banking information contained on the page.

Luckily, the SOP will prevent the malicious script hosted on *attacker.com* from reading the HTML data returned from *onlinebank.com*. This keeps the malicious script on page A from obtaining sensitive information embedded within page B.

Learn to Program

You should now have a solid background to help you understand most of the vulnerabilities we will cover. Before you set up your hacking tools, I recommend that you learn to program. Programming skills are helpful, because hunting for bugs involves many repetitive tasks, and by learning a programming language such as Python or shell scripting, you can automate these tasks to save yourself a lot of time.

You should also learn to read JavaScript, the language with which most sites are written. Reading the JavaScript of a site can teach you about how it works, giving you a fast track to finding bugs. Many top hackers say that their secret sauce is that they read JavaScript and search for hidden endpoints, insecure programming logic, and secret keys. I've also found many vulnerabilities by reading JavaScript source code.

Codecademy is a good resource for learning how to program. If you prefer to read a book instead, *Learn Python the Hard Way* by Zed Shaw (Addison-Wesley Professional, 2013) is a great way to learn Python. And *Eloquent JavaScript*, Third Edition, by Marijn Haverbeke (No Starch Press, 2019) is one of the best ways to master JavaScript.

4

ENVIRONMENTAL SETUP AND TRAFFIC INTERCEPTION

You'll save yourself a lot of time and headache if you hunt for bugs within a well-oiled lab. In this chapter, I'll guide you, step-by-step, through setting up your hacking environment. You'll configure your browser to work with Burp Suite, a web proxy that lets you view and alter HTTP requests and responses sent between your browser and web servers. You'll learn to use Burp's features to intercept web traffic, send automated and repeated requests, decode encoded content, and compare requests. I will also talk about how to take good bug bounty notes.

This chapter focuses on setting up an environment for web hacking only. If your goal is to attack mobile apps, you'll need additional setup and tools. We'll cover these in Chapter 23, which discusses mobile hacking.

Choosing an Operating System

Before we go on, the first thing you need to do is to choose an operating system. Your operating system will limit the hacking tools available to you. I recommend using a Unix-based system, like Kali Linux or macOS, because many open source hacking tools are written for these systems. *Kali Linux* is a Linux distribution designed for digital forensics and hacking. It includes many useful bug bounty tools, such as Burp Suite, recon tools like DirBuster and Gobuster, and fuzzers like Wfuzz. You can download Kali Linux from *https://www.kali.org/downloads/*.

If these options are not available to you, feel free to use other operating systems for hacking. Just keep in mind that you might have to learn to use different tools than the ones mentioned in this book.

Setting Up the Essentials: A Browser and a Proxy

Next, you need a web browser and a web proxy. You'll use the browser to examine the features of a target application. I recommend using Firefox, since it's the simplest to set up with a proxy. You can also use two different browsers when hacking: one for browsing the target, and one for researching vulnerabilities on the internet. This way, you can easily isolate the traffic of your target application for further examination.

A *proxy* is software that sits between a client and a server; in this case, it sits between your browser and the web servers you interact with. It intercepts your requests before passing them to the server, and intercepts the server's responses before passing them to you, like this:

Browser <----------> Proxy <----------> Server

Using a proxy is essential in bug bounty hunting. Proxies enable you to view and modify the requests going out to the server and the responses coming into your browser, as I'll explain later in this chapter. Without a proxy, the browser and the server would exchange messages automatically, without your knowledge, and the only thing you would see is the final resulting web page. A proxy will instead capture all messages before they travel to their intended recipient.

Proxies therefore allow you to perform recon by examining and analyzing the traffic going to and from the server. They also let you examine interesting requests to look for potential vulnerabilities and exploit these vulnerabilities by tampering with requests.

For example, let's say that you visit your email inbox and intercept the request that will return your email with a proxy. It's a GET request to a URL that contains your user ID. You also notice that a cookie with your user ID is included in the request:

```
GET /emails/USER_ID HTTP/1.1
Host: example.com
Cookie: user_id=USER_ID
```

In this case, you can try to change the `USER_ID` in the URL and the `Cookie` header to another user's ID and see if you can access another user's email.

Two proxies are particularly popular with bug bounty hunters: Burp Suite and the Zed Attack Proxy (ZAP). This section will show you how to set up Burp, but you're free to use ZAP instead.

Opening the Embedded Browser

Both Burp Suite and ZAP come with embedded browsers. If you choose to use these embedded browsers for testing, you can skip the next two steps. To use Burp Suite's embedded browser, click **Open Browser** in Burp's Proxy tab after it's launched (Figure 4-1). This embedded browser's traffic will be automatically routed through Burp without any additional setup.

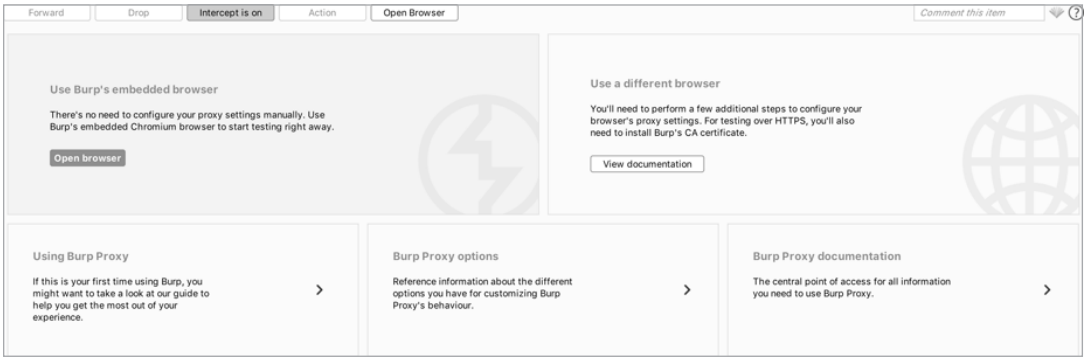


Figure 4-1: You can use Burp's embedded browser instead of your own external browser for testing.

Setting Up Firefox

Burp's embedded browser offers a convenient way to start bug hunting with minimal setup. However, if you are like me and prefer to test with a browser you are used to, you can set up Burp to work with your browser. Let's set up Burp to work with Firefox.

Start by downloading and installing your browser and proxy. You can download the Firefox browser from *https://www.mozilla.org/firefox/new/* and Burp Suite from *https://portswigger.net/burp/*.

Bug bounty hunters use one of two versions of Burp Suite: Professional or Community. You have to purchase a license to use Burp Suite Professional, while the Community version is free of charge. Burp Suite Pro includes a vulnerability scanner and other convenient features like the option to save a work session to resume later. It also offers a full version of the Burp Intruder, while the Community version includes only a limited version. In this book, I cover how to use the Community version to hunt for bugs.

Now you have to configure your browser to route traffic through your proxy. This section teaches you how to configure Firefox to work with Burp Suite. If you're using another browser-proxy combination, please look up their official documentation for tutorials instead.

Launch Firefox. Then open the Connections Settings page by choosing **Preferences ▸ General ▸ Network Settings**. You can access the Preferences tab from the menu at Firefox's top-right corner (Figure 4-2).

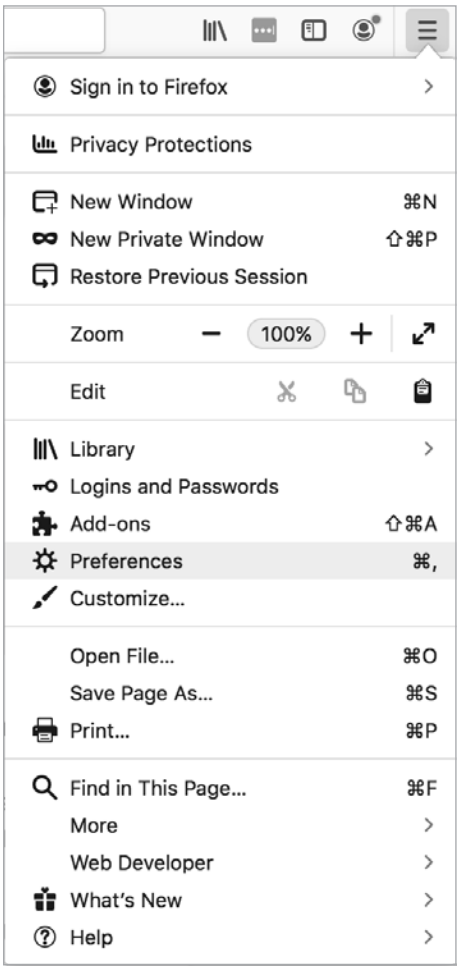


Figure 4-2: You can find the Preferences option at the top-right corner of Firefox.

The Connection Settings page should look like the one in Figure 4-3.

Select **Manual proxy configuration** and enter the IP address **127.0.0.1** and port **8080** for all the protocol types. This will tell Firefox to use the service running on port 8080 on our machine as a proxy for all of its traffic. 127.0.0.1 is the localhost IP address. It identifies your current computer, so you can use it to access the network services running on your machine. Since Burp runs on port 8080 by default, this setting tells Firefox to route all traffic through Burp. Click **OK** to finalize the setting. Now Firefox will route all traffic through Burp.

Figure 4-3: Configure Firefox's proxy settings on the Connection Settings page.

Setting Up Burp

After downloading Burp Suite, open it and click **Next**, then **Start Burp**. You should see a window like Figure 4-4.

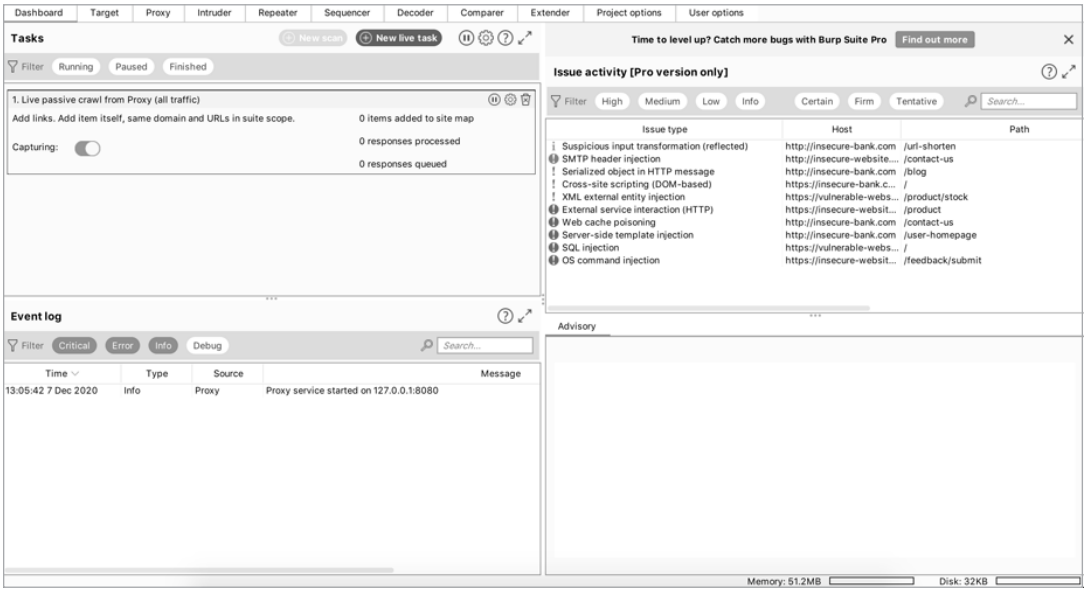


Figure 4-4: Burp Suite Community Edition startup window

Now let's configure Burp so it can work with HTTPS traffic. Remember that HTTPS protects your data's privacy by encrypting your traffic, making sure only the two parties in a communication (your browser and the server) can decrypt it. This also means your Burp proxy won't be able to intercept HTTPS traffic going to and from your browser. To work around this issue, you need to show Firefox that your Burp proxy is a trusted party by installing its certificate authority (CA) certificate.

Let's install Burp's certificate on Firefox so you can work with HTTPS traffic. With Burp open and running, and your proxy settings set to 127.0.0.1:8080, go to *http://burp/* in your browser. You should see a Burp welcome page (Figure 4-5). Click **CA Certificate** at the top right to download the certificate file; then click **Save File** to save it in a safe location.

Figure 4-5: Go to http://burp/ to download Burp's CA certificate.

Next, in Firefox, click **Preferences** ▸ **Privacy & Security** ▸ **Certificates** ▸ **View Certificates** ▸ **Authorities**. Click **Import** and select the file you just saved, and then click **Open**. Follow the dialog's instructions to trust the certificate to identify websites (Figure 4-6).

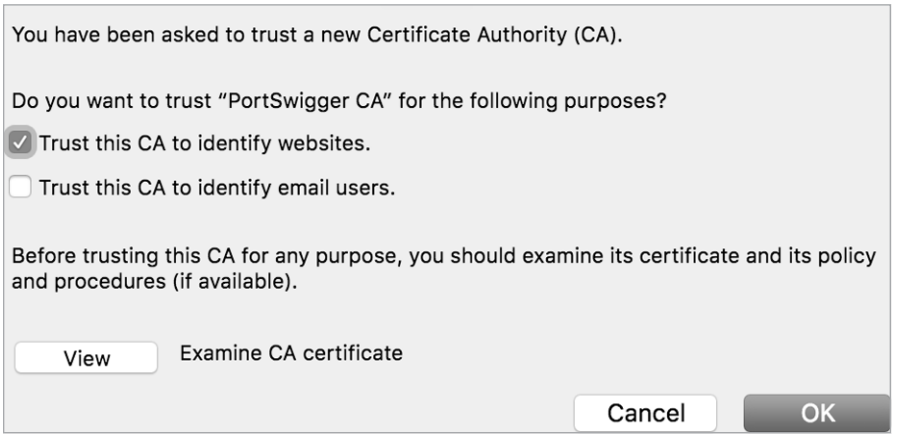


Figure 4-6: Select the Trust this CA to identify websites option in Firefox's dialog.

Restart Firefox. Now you should be all set to intercept both HTTP and HTTPS traffic.

Let's perform a test to make sure that Burp is working properly. Switch to the Proxy tab in Burp and turn on traffic interception by clicking **Intercept Is Off**. The button should now read Intercept Is On (Figure 4-7). This means you're now intercepting traffic from Firefox or the embedded browser.

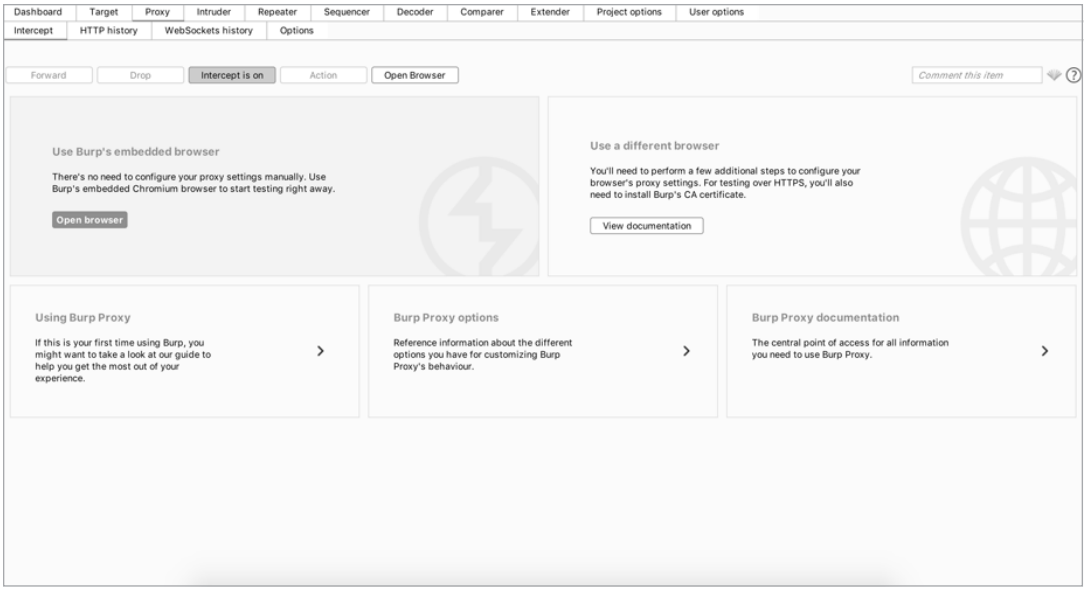


Figure 4-7: Intercept Is On means that you're now intercepting traffic.

Then open Firefox and visit *https://www.google.com*. In Burp's proxy, you should see the main window starting to populate with individual requests. The Forward button in Burp Proxy will send the current request to the designated server. Click **Forward** until you see the request with the hostname *www.google.com*. If you see this request, Burp is correctly intercepting Firefox's traffic. It should begin like this:

```
GET / HTTP/1.1
Host: www.google.com
```

Click **Forward** to send the request over to Google's server. You should see Google's home page appear in your Firefox window.

If you aren't seeing requests in Burp's window, you might not have installed Burp's CA certificate properly. Follow the steps in this chapter to reinstall the certificate. In addition, check that you've set the correct proxy settings to 127.0.0.1:8080 in Firefox's Connection Settings.

Using Burp

Burp Suite has a variety of useful features besides the web proxy. Burp Suite also includes an *intruder* for automating attacks, a *repeater* for manipulating individual requests, a *decoder* for decoding encoded content, and a *comparer* tool for comparing requests and responses. Of all Burp's features, these are the most useful for bug bounty hunting, so we'll explore them here.

The Proxy

Let's see how you can use the Burp *proxy* to examine requests, modify them, and forward them to Burp's other modules. Open Burp and switch to the Proxy tab, and start exploring what it does! To begin intercepting traffic, make sure the Intercept button reads Intercept Is On.

When you browse to a site on Firefox or Burp's embedded browser, you should see an HTTP/HTTPS request appear in the main window. When intercept is turned on, every request your browser sends will go through Burp, which won't send them to the server unless you click Forward in the proxy window. You can use this opportunity to modify the request before sending it to the server or to forward it over to other modules in Burp. You can also use the search bar at the bottom of the window to search for strings in the requests or responses.

To forward the request to another Burp module, right-click the request and select **Send to *Module*** (Figure 4-8).

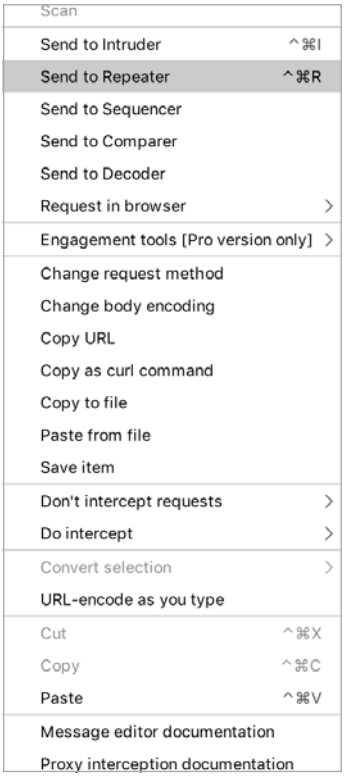


Figure 4-8: You can forward the request or response to different Burp modules by right-clicking it.

Let's practice intercepting and modifying traffic by using Burp Proxy! Go to Burp Proxy and turn on traffic interception. Then open Firefox or Burp's embedded browser and visit the URL *https://www.google.com*. As you did in the preceding section, click **Forward** until you see the request with the hostname *www.google.com*. You should see a request like this one:

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0
Accept-Language: en-US
Accept-Encoding: gzip, deflate
Connection: close
```

Let's modify this request before sending it. Change the `Accept-Language` header value to **`de`**.

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0
Accept-Language: de
Accept-Encoding: gzip, deflate
Connection: close
```

Click **Forward** to send the request over to Google's server. You should see Google's home page in German appear in your browser's window (Figure 4-9).

Figure 4-9: Google's home page in German

If you're a German speaker, you could do the test in reverse: switch the `Accept-Language` header value from `de` to `en`. You should see the Google home page in English. Congratulations! You've now successfully intercepted, modified, and forwarded an HTTP request via a proxy!

The Intruder

The Burp *intruder* tool automates request sending. If you are using the Community version of Burp, your intruder will be a limited, trail version. Still, it allows you to perform attacks like *brute-forcing*, whereby an attacker submits many requests to a server using a list of predetermined values and sees if the server responds differently. For example, a hacker who obtains a list of commonly used passwords can try to break into your account by repeatedly submitting login requests with all the common passwords. You can send requests over to the intruder by right-clicking a request in the proxy window and selecting **Send to intruder**.

The **Target** screen in the intruder tab lets you specify the host and port to attack (Figure 4-10). If you forward a request from the proxy, the host and port will be prefilled for you.

Figure 4-10: You can specify the host and port to attack on the Target screen.

The intruder gives several ways to customize your attack. For each request, you can choose the payload and payloads positions to use. The *payloads* are the data that you want to insert into specific positions in the request. The *payload positions* specify which parts of the request will be replaced by the payloads you choose. For example, let's say users log into *example.com* by sending a POST request to *example.com/login*. In Burp, this request might look like this:

```
POST /login HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-US
Accept-Encoding: gzip, deflate
Connection: close
username=vickie&password=abc123
```

The POST request body contains two parameters: `username` and `password`. If you were trying to brute-force a user's account, you could switch up the `password` field of the request and keep everything else the same. To do that, specify the payload positions in the **Positions** screen (Figure 4-11). To add a portion of the request to the payload positions, highlight the text and click **Add** on the right.

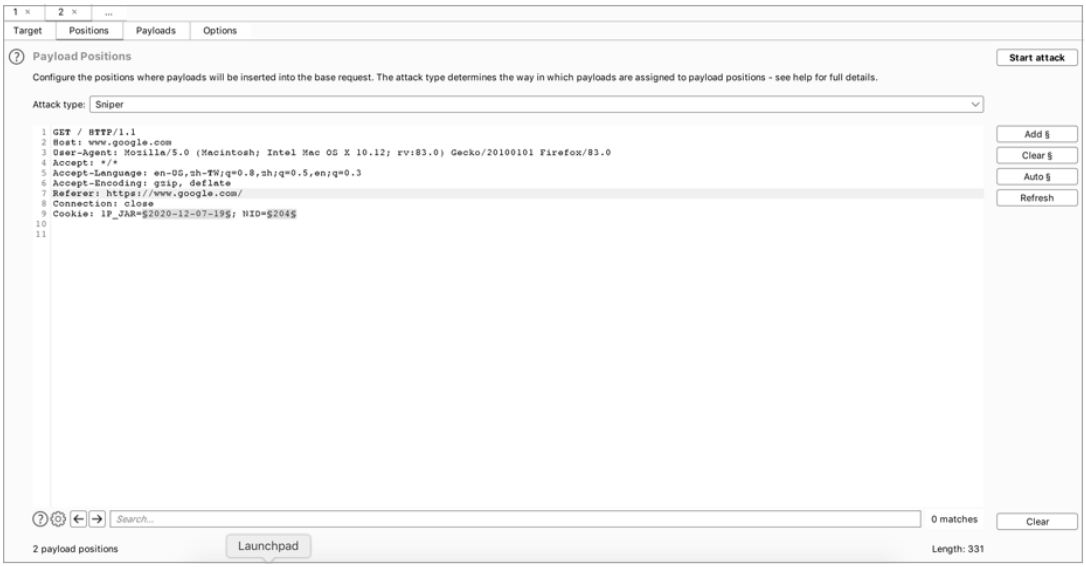


Figure 4-11: You can specify the payload positions in the Positions screen.

Then, switch over to the **Payloads** screen (Figure 4-12). Here, you can choose payloads to insert into the request. To brute-force a login password, you can add a list of commonly used passwords here. You can also, for example, use a list of numbers with which to brute-force IDs in requests, or use an attack payload list you downloaded from the internet. Reusing attack payloads shared by others can help you find bugs faster. We will talk more about how to use reused payloads to hunt for vulnerabilities in Chapter 25.

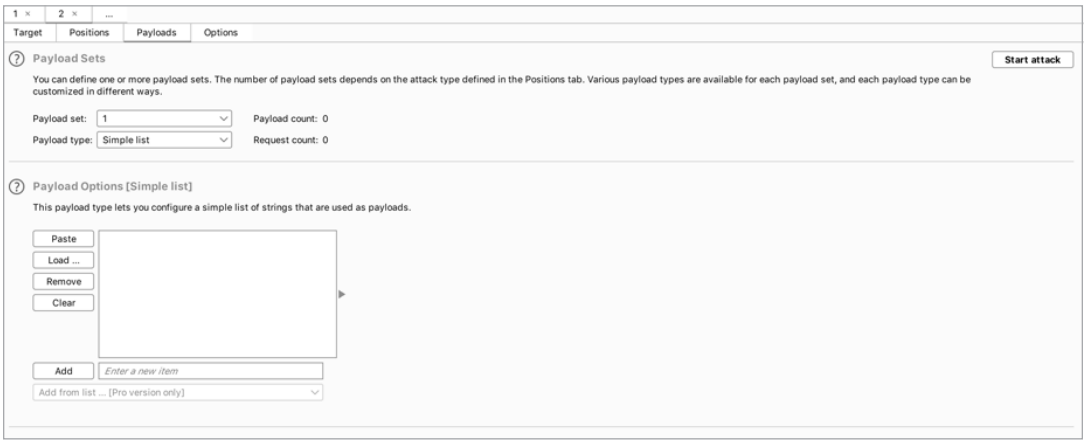


Figure 4-12: Choose your payload list on the Payloads screen.

Once you've specified those, click the **Start Attack** button to start the automated test. The intruder will send a request for each payload you listed and record all responses. You can then review the responses and response codes and look for interesting results.

The Repeater

The *repeater* is probably the tool you'll use the most often (Figure 4-13). You can use it to modify requests and examine server responses in detail. You could also use it to bookmark interesting requests to go back to later.

Although the repeater and intruder both allow you to manipulate requests, the two tools serve very different purposes. The intruder automates attacks by automatically sending programmatically modified requests. The repeater is meant for manual, detailed modifications of a single request.

Send requests to the repeater by right-clicking the request and selecting **Send to repeater**.

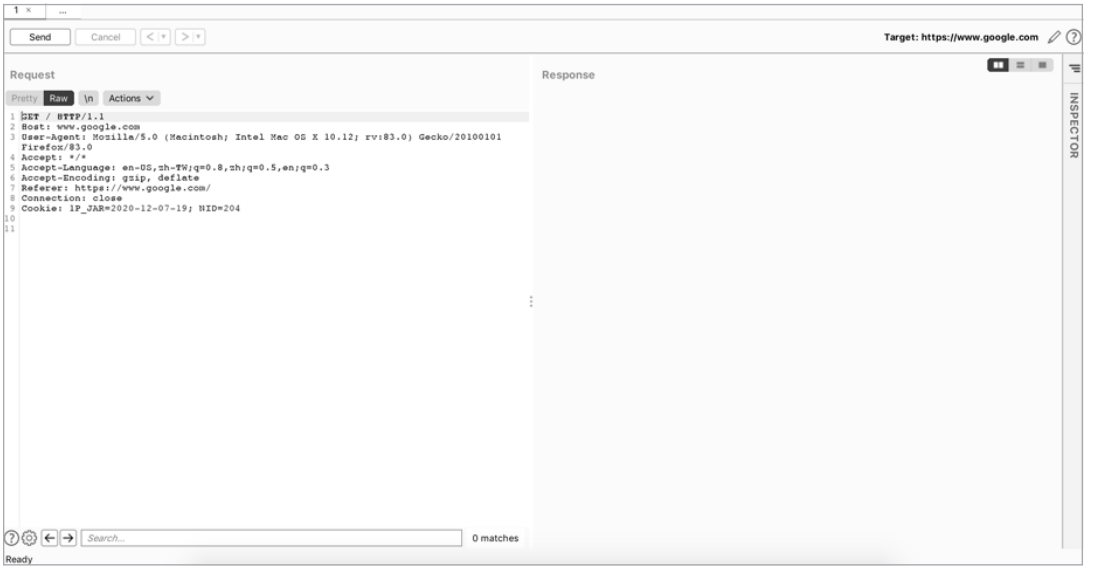


Figure 4-13: The repeater is good for close examination of requests and manual exploitation.

On the left of the repeater screen are requests. You can modify a request here and send the modified request to the server by clicking **Send** at the top. The corresponding response from the server will appear on the right.

The repeater is good for exploiting bugs manually, trying to bypass filters, and testing out different attack methods that target the same endpoint.

The Decoder

The Burp *decoder* is a convenient way to encode and decode data you find in requests and responses (Figure 4-14). Most often, I use it to decode, manipulate, and re-encode application data before forwarding it to applications.

Figure 4-14: You can use the decoder to decode application data to read or manipulate its plaintext.

Send data to the decoder by highlighting a block of text in any request or response, then right-clicking it and selecting **Send to decoder**. Use the drop-down menus on the right to specify the algorithm to use to encode or decode the message. If you're not sure which algorithm the message is encoded with, try to **Smart Decode** it. Burp will try to detect the encoding, and decode the message accordingly.

The Comparer

The *comparer* is a way to compare requests or responses (Figure 4-15). It highlights the differences between two blocks of text. You might use it to examine how a difference in parameters impacts the response you get from the server, for example.

Send data over to the comparer by highlighting a block of text in any request or response, then right-clicking it and selecting **Send to comparer**.

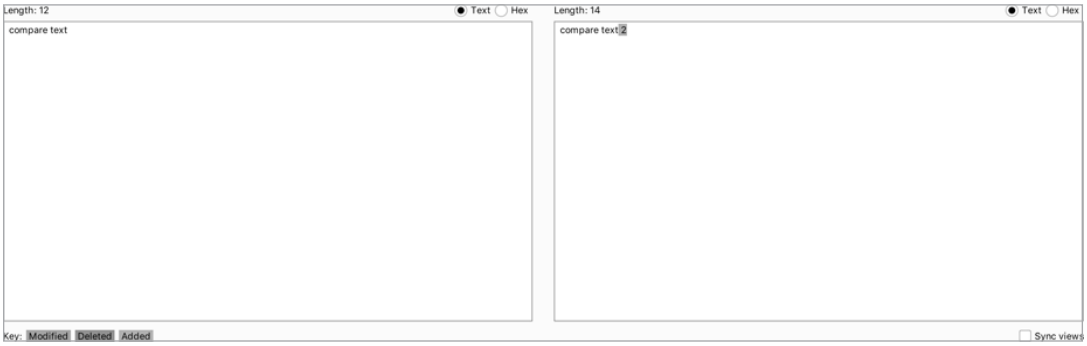


Figure 4-15: The comparer will highlight the differences between two blocks of text.

Saving Burp Requests

You can save requests and responses on Burp as well. Simply right-click any request and select **Copy URL**, **Copy as curl command**, or **Copy to file** to store these results into your note folder for that target. The Copy URL option copies the URL of the request. Copy as curl command copies the entire request, including the request method, URL, headers, and body as a curl command. And Copy to file saves the entire request to a separate file.

A Final Note on . . . Taking Notes

Before you get started looking for vulnerabilities in the next chapter, a quick word of advice: organizational skills are critical if you want to succeed in bug bounties. When you work on targets with large scopes or hack multiple targets at the same time, the information you gather from the targets could balloon and become hard to manage.

Often, you won't be able to find bugs right away. Instead, you'll spot a lot of weird behaviors and misconfigurations that aren't exploitable at the moment but that you could combine with other behavior in an attack later on. You'll need to take good notes about any new features, misconfigurations, minor bugs, and suspicious endpoints that you find so you can quickly go back and use them.

Notes also help you plan attacks. You can keep track of your hacking progress, the features you've tested, and those you still have to check. This prevents you from wasting time by testing the same features over and over again.

Another good use of notes is to jot down information about the vulnerabilities you learn about. Record details about each vulnerability, such as its theoretical concept, potential impact, exploitation steps, and sample proof-of-concept code. Over time, this will strengthen your technical skills and build up a technique repository that you can revisit if needed.

Since these notes tend to balloon in volume and become very disorganized, it's good to keep them organized from the get-go. I like to take notes in plaintext files by using Sublime Text (*https://www.sublimetext.com/*) and organize them by sorting them into directories, with subdirectories for each target and each topic.

For example, you can create a folder for each target you're working on, like Facebook, Google, or Verizon. Then, within each of these folders, create files to document interesting endpoints, new and hidden features, reconnaissance results, draft reports, and POCs.

Find a note-taking and organizational strategy that works for you. For example, if you are like me and prefer to store notes in plaintext, you can search around for an integrated development environment (IDE) or text editor that you feel the most comfortable in. Some prefer to take notes using the Markdown format. In this case, Obsidian (*https://obsidian.md/*) is an excellent tool that displays your notes in an organized way. If you like to use mind maps to organize your ideas, you can try the mind-mapping tool XMind (*https://www.xmind.net/*).

Keep your bug bounty notes in a centralized place, such as an external hard drive or cloud storage service like Google Drive or Dropbox, and don't forget to back up your notes regularly!

In summary, here are a few tips to help you take good notes:

- Take notes about any weird behaviors, new features, misconfigurations, minor bugs, and suspicious endpoints to keep track of potential vulnerabilities.
- Takes notes to keep track of your hacking progress, the features you've tested, and those you still have to check.

- Take notes while you learn: jot down information about the vulnerabilities you learn about, like its theoretical concept, potential impact, exploitation steps, and sample POC code.
- Keep your notes organized from the get-go, so you can find them when you need to!
- Find a note-taking and organizational process that works for you. You can try out note-taking tools like Sublime Text, Obsidian, and XMind to find a tool that you prefer.